

PROGRAMACION BASICA

Jonathan Pearce y
Graham Bland

PARANINFO

MSX

Programación básica



GRAHAM BLAND y JONATHAN PEARCE

MSX
Programación básica

SEGUNDA EDICION

1986



MADRID

Traducido por:
FELISA MATEO GARCIA

Título original inglés:
MSX an introduction
publicado por:
Century Communications Ltd,
London W1V 5LE

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 0 7126 0538 X (Edición inglesa)
ISBN: 84-283-1407-1 (Edición española)

Depósito Legal: M-44585-1985



Magallanes, 25 -28015 Madrid (01301/3669)

ALCO, artes gráficas. Jaspe, 34 - 28026 MADRID

Indice de materias

Prólogo

La introducción del estándar MSX es, sin lugar a dudas, el hecho más importante en la historia de la computerización en el hogar. Verdaderamente sus implicaciones van más allá que las de las máquinas individuales incluso de algunas marcadamente populares como el Sinclair ZX Spectrum.

Hasta ahora el mercado de los ordenadores domésticos o personales ha estado altamente fragmentado, compuesto de un número de máquinas total o parcialmente incompatibles. Los juegos y otros paquetes de software escritos para ser utilizados en una máquina, no funcionan en otra. Del mismo modo, los mandos de juego y otros accesorios diseñados para una máquina no se pueden conectar a otrá. Esto representa una dificultad innecesaria para el usuario y encarece todos los aspectos. Aunque existe una cantidad importante de software para los ordenadores caseros, está repartido más o menos uniformemente entre un grupo considerable de máquinas, de manera que en una de ellas sólo funciona una parte de los paquetes disponibles. Incluso con la máquina más popular, es posible que el usuario no sea capaz de encontrar un paquete que se ajuste perfectamente a sus necesidades.

El MSX marca un final para los problemas de este tipo porque define una especificación del hardware común que debe estar incluida en todas las máquinas MSX, y un lenguaje común –MSX-BASIC– en el que se pueden escribir los paquetes de aplicaciones y juegos. El cliente tiene la garantía de que cualquiera que sea la máquina MSX que elija, será capaz de ejecutar y procesar, perfectamente, cualquier paquete software MSX. Esto facilita mucho las cosas a los creadores de software ya que no tienen que volver a escribir y reorganizar sus programas para cada una de las múltiples máquinas similares, pero incompatibles. El usuario obtiene aún mayores beneficios, ya que de esta forma tiene acceso a un número incomparable de paquetes de software, los cuales resultan relativamente baratos debido a las cantidades involucradas.

Así pues la teoría que respalda al MSX es una teoría sólida y proporcionará enormes beneficios a los usuarios de los ordenadores. No es, en absoluto, ninguna exageración afirmar que la llegada del estándar

MSX será reconocida como un hito en el campo de los ordenadores universales.

También resulta interesante conocer el motivo que dió lugar al MSX y las compañías que lo han adoptado. Al contrario que en América y en el Reino Unido, el mercado de los ordenadores domésticos en Japón ha estado relativamente poco desarrollado. En la primavera de 1983, NEC y Matsushita establecieron contactos con Microsoft, uno de los suministradores independientes de software para micro-ordenadores más importante del mundo, con la especificación para un ordenador personal que iban a desarrollar conjuntamente. Querían que Microsoft les escribiera, para la máquina, una versión de su BASIC estándar para la industria, con vistas a desarrollar las ventas en el mercado nacional japonés. En junio, otros 12 fabricantes, incluida una compañía americana, hacía a Microsoft la misma petición.

Para evitar tener que escribir 14 versiones distintas del BASIC perpetuando así los problemas de compatibilidad del mercado de micros caseros, Microsoft tuvo la idea del estándar MSX. Esto incluye chips del procesador generalmente disponibles y, por supuesto, una versión optimizada y ampliada del intérprete BASIC de Microsoft. El 16 de junio de 1983, Microsoft anunció el MSX, así como las compañías que iban a adoptarlo: Canon, Fujitsu, General, Hitachi, JVC, Kyocera, Matsushita, Mitsubishi, NEC, Pioneer, Sanyo, Sony, Spectravideo (US), Toshiba y Yamaha —todas ellas grandes compañías establecidas en los mercados consumidores de componentes electrónicos.

Lo que este anuncio significaba era que toda la industria japonesa de los ordenadores y de los componentes electrónicos se había reunido para producir ordenadores domésticos bajo una única especificación. Los primeros frutos aparecieron en la Feria de la Electrónica de Japón, en Osaka en octubre de 1983, cuando la posibilidad de intercambiar cartuchos de juegos entre las máquinas MSX de Hitachi, JVC, Matsushita, Mitsubishi, National, NEC, Sanyo, Sony y Toshiba causó cierta sensación.

La absoluta firmeza con que los japoneses comercializan sus productos y los volúmenes que son capaces de producir (la producción inicial de máquinas MSX era de 53.000 unidades al mes) está imponiendo la estandarización en la industria del área de pequeños ordenadores, de la misma forma que las cintas de casete revolucionaron el mercado del sonido en los primeros años 70.

Dada la enorme rapidez con que está creciendo el MSX, obviamente resulta difícil prever qué es lo que va a ocurrir a continuación. Sin embargo, el proyecto incluye dos piedras angulares que permiten efectuar algún tipo de especulación.

En primer lugar, el 5 de octubre de 1983, Microsoft anunció el MSX-DOS, un sistema operativo de 8 bits diseñado específicamente para las máquinas MSX. En pocas palabras, el MSX-DOS no sólo permite a las máquinas MSX soportar el almacenamiento en disco (incrementando así enormemente la cantidad de datos que pueden albergar), también las hace compatibles, en un nivel superior, con los sistemas operativos MS-DOS y XENIX, ambos utilizados por micros más potentes y por minis.

Lisa y llanamente, esto significa que un usuario podrá trabajar en su oficina con aplicaciones como hojas de cálculo o procesadores de texto y, al final de la jornada, retirar el disquete que contiene todos los datos y llevárselo a casa para su terminación en una máquina MSX. ¡Ya no hay que quedarse hasta la noche en la oficina, ni duplicar el trabajo!

En segundo lugar, el BASIC suministrado con MSX se puede ampliar para controlar un número prácticamente ilimitado de "añadidos". Esto incluye dispositivos estándares como grabadores de cintas, impresoras, mandos de juego, paletas de juego y teclados especiales de bolsillo así como dispositivos especiales como sintetizadores de voz, sintetizadores de FM y discos de vídeo. Ya han aparecido máquinas que manejan gráficos y vídeo compuesto, grabadores y cámaras de vídeo, lápices luminosos y brazos mecánicos. Se ha diseñado una máquina Yamaha para enseñar al usuario a manejar un sintetizador por lo que trae incorporado un teclado completo de sintetizador.

Las primeras máquinas Sanyo incorporan una posibilidad conocida como "captadores de imagen" ("Frame grabbing"), que le permite almacenar una imagen de vídeo o TV y modificarla posteriormente añadiendo sus propios gráficos. Se oyen rumores sobre MSX hi-fi y en principio no existe ninguna razón por la que cualquier equipo electrónico doméstico no pueda ser compatible con el MSX y controlado por un ordenador MSX central. Así pues el MSX abre las vías de utilización de los ordenadores caseros en una serie de aplicaciones a las que los ordenadores ni siquiera se habían acercado. Es importante, que cuando lea este libro, tenga esto presente porque las técnicas de programación que vamos a presentar en las páginas siguientes le permitirán enseguida hacer muchas más cosas además de calcular el interés compuesto de la cuenta de su sociedad constructora o el gasto de su coche.

Introducción

"Introducción al MSX" es un libro de iniciación para los nuevos usuarios del MSX. Confiamos que le enseñará todo lo que necesita saber para escribir programas útiles y divertidos en MSX-BASIC, lo cual no quiere decir que Vd. ya no va a adquirir más libros sobre MSX —todo el mundo compra libros que traen listados de programas que hacen cosas específicas, y a medida que vaya Vd. avanzando, es posible que le interese aprender el código máquina del MSX, el sistema operativo en disco del MSX y así sucesivamente.

Hemos diseñado este libro para conducirle directamente desde los principios más elementales de los programas —los nombres de los mandatos utilizados en MSX-BASIC así como sus significados exactos y posibilidades— hasta las complejidades de la programación en BASIC avanzada pasando por los conceptos que subyacen en la estructura de los programas. También hemos incluido unos capítulos dedicados específicamente a gráficos y música. Ambas funciones tienen lenguajes dedicados, independientes y fáciles de comprender, a los que se puede acceder desde el MSX-BASIC y que le permitirán escribir sus propios programas de juegos y sus tonadas.

Sin embargo, empezaremos el libro haciéndole una presentación de los ordenadores en general y de los ordenadores MSX en particular. Por el prefacio ya habrá comprendido que el MSX marca una nueva etapa para los ordenadores personales y caseros. Con el capítulo siguiente le introduciremos en la terminología que vamos a emplear para describir los ordenadores y le hablaremos sobre los aspectos más elementales del hardware, los periféricos y el software.

Esperamos que Vd. disfrute leyendo este libro, tanto como nosotros escribiéndolo y que encuentre provechoso el tiempo que le haya dedicado.

Jonathan Pearce
Graham Bland
Mark Adams
Tom Lewis
Reflex Communications Ltd.

Presentación del MSX

Cualquiera que sea su grado de interés en los ordenadores caseros es posible que Vd. ya conozca algo sobre cómo trabajan o por lo menos habrá oído algo de la jerga asociada a los ordenadores. Nuestra intención, en este capítulo, es presentarles las palabras más comúnmente empleadas y explicarles exactamente qué diferencia existe entre su ordenador MSX y las generaciones anteriores de ordenadores caseros.

Empezaremos echando una mirada a los distintos tipos de ordenadores y las aplicaciones para las que son más comúnmente utilizados para seguir después con los elementos que configuran todos los sistemas de ordenador: el propio **hardware** del ordenador, los **periféricos** que se le pueden conectar, y el **software** que funciona en ellos.

Micros, minis y grandes ordenadores

Sin ningún lugar a dudas, ha sido el advenimiento del **microordenador** lo que ha promocionado más que ninguna otra cosa, la aceptación masiva de los ordenadores. Es más, su impacto ha sido tan fuerte que hoy en día se considera ampliamente aceptado el hecho de que ser un "entendido en ordenadores" es uno de los requerimientos básicos para la vida moderna, ya que si uno no es capaz de manejar los conceptos básicos de los ordenadores, ya puede darse por vencido.

Mientras que los primeros ordenadores caseros eran, en realidad, algo más que calculadoras ampliadas, ahora, han alcanzado un alto nivel de sofisticación y un precio que prácticamente todo el mundo puede afrontar. Es ahora cuando, millones de personas, en todo el mundo, están empezando a darles una utilización real, en lugar de emplearlos sólo para jugar. Los primeros ordenadores MSX, por ejemplo, le permiten no sólo practicar con juegos y realizar cálculos elementales, también incluye unas facilidades especiales para enseñarle a tocar el teclado de un sintetizador o para "capturar" una imagen desde su televisor y añadirle los gráficos que Vd. haya generado. Este tipo de posibilidades eran inimaginables en los ordenadores caseros de hace apenas un año, de manera que es fácil ver lo que representa el salto cuántico del MSX (con todos los respetos para Sir Clive Sinclair).

No podemos olvidar, sin embargo, que los ordenadores caseros no son más que una pequeña parte de la industria de los microordenadores. Los microordenadores están haciendo sentir su presencia en el mundo de los negocios, donde se dedican a tareas tales como el proceso de textos, planificación financiera, contabilidad y control de almacenes.

Subiendo un peldaño por encima de los micros, llegamos a los **mini-ordenadores**. Fueron muy populares en los años 70 y a principios de los 80 ya que se les consideraba como una alternativa práctica de los **grandes ordenadores**. Los grandes ordenadores eran los enormes ordenadores utilizados principalmente por las grandes empresas para almacenar cantidades masivas de información. Los mini-ordenadores se sitúan entre los micros y los grandes ordenadores. Su popularidad creció rápidamente al permitir que los usuarios almacenasen información en un conjunto de sistemas conectados por líneas telefónicas, en lugar de obligarles a depender de la instalación enorme pero única de un gran ordenador (también conocido como ordenador central y ordenador principal). Además, la potencia de los mini-ordenadores ha ido aproximándose rápidamente a la de los grandes ordenadores proporcionando, por lo tanto, una base realmente descentralizada para el almacenamiento de la información.

Recientemente, sin embargo, el incremento de la potencia de los grandes micro-ordenadores ha hecho a los mini-ordenadores lo que éstos hicieron a los grandes hace unos pocos años. En otras palabras, los micro-ordenadores han ido adquiriendo progresivamente una mayor potencia mientras bajaba su precio y su tamaño.

Un fabricante llamado Hewlett-Packard, ha presentado lo que llama "Un ordenador grande en su escritorio". Los límites tradicionales entre los diferentes ordenadores se difuminan cada vez más y los micros se están convirtiendo en una parte mucho más importante de la industria hasta el extremo de que si Vd. puede aprender cómo utilizar un micro, posiblemente no necesitará ir más allá.

Por último, debemos mencionar los llamados **super-ordenadores**. Son los ordenadores que Vd. ha visto en las películas como *2001 Odisea en el espacio* o *El Juego de la Guerra*. Antes de la llegada de los ordenadores caseros, eran ordenadores como éstos los que compendiaban la visión que el público tenía de los ordenadores-potentes, amenazadores y totalmente incomprensibles. Con la lectura de este libro, esperamos que Vd. llegue a compartir nuestra visión de los ordenadores —que normalmente son pequeños, amenos y hoy en día muy fáciles de utilizar.

Habiendo dado una pasada por los diferentes tipos de ordenadores y sus diversas utilizaciones, vamos ahora a considerar las partes que les configuran. En términos de su estructura y de la forma en que funcio-

nan juntos sus componentes, su ordenador MSX es similar a otras máquinas de tamaño mucho mayor. Aunque otras máquinas puedan hacer las cosas de formas ligeramente diferentes, lo esencial que hay dentro de ellas, y su modo de operar, son idénticos. Tomaremos el MSX como base para presentarle las siguientes nociones y empezaremos con la caja que acaba de comprar –"el hardware".

Hardware

Resulta difícil pensar en alguien que lea este libro y no haya oído hablar de los **microchips**. Aparecen por donde quiera que mire, desde los tostadores y hornos de microondas a los televisores y vídeos. ¡Están incluso en los coches para permitirles dirigir advertencias a los conductores! Es pues difícil pensar en un sólo dispositivo electro-mecánico que no incluya un microchip.

Su ordenador MSX tiene tres "microchips" o **micropocesadores** principales, cada uno de los cuales realiza una función específica. El primero y más importante es la **unidad central de proceso** o **CPU** (del inglés **central processing unit**) que en las máquinas MSX es un Zilog Z80A (sin lugar a dudas el chip procesador más extendido para los ordenadores caseros en estos momentos). El procesador central es, digámoslo así, el motor que dirige al ordenador. Se encarga de todas las funciones matemáticas y de los cálculos necesarios para que el ordenador funcione e incluso recibe sus instrucciones en su propio lenguaje. Este lenguaje se llama **código máquina**. Independientemente del lenguaje que Vd. utilice para dirigirse al ordenador –BASIC, LOGO, Lenguaje Ensamblador o cualquier otro– al final siempre se traduce al código máquina de la CPU. Los lenguajes como el BASIC necesitan un proceso importante de traducción y ésto lleva tiempo. Otros lenguajes, como el Ensamblador, necesitan poca traducción y por lo tanto se ejecutan más rápidamente que el BASIC. Sin embargo, programar en esos lenguajes resulta difícil y engorroso. Cualquier juego o cualquier programa sofisticado que Vd. compre para su ordenador probablemente está escrito en código máquina.

El segundo chip principal es el Texas Instruments TMS 9918A procesador de gráficos. El ordenador lo utiliza para controlar todo lo que va a aparecer en su pantalla de televisión. Muchos ordenadores caseros tienen un único procesador que hace todas estas cosas. En los ordenadores MSX, el Z80 central se utiliza para procesar y controlar solamente el teclado (de esta forma se entera de cuándo quiere Vd. hacer algo). El procesador de gráficos y el chip de sonido General Instruments AY-3-8910 controlan el resto. Es por esto por lo que los ordenadores

MSX ofrecen unas posibilidades gráficas y musicales excelentes, como demostraremos más adelante. En las secciones de los Macro Lenguajes para los gráficos y la música, hemos incluido numerosos programas que muestran exactamente cómo poner en funcionamiento estas posibilidades.

Memoria

Siguiendo con la potencia de proceso de su ordenador, Vd. habrá visto en el libro de instrucciones, que también incluye un mínimo de 32K de **RAM** y 32K de **ROM**. Pero ¿qué significan estas 32K de RAM y ROM?

Para que los procesadores de su ordenador MSX puedan trabajar es preciso que reciban información de alguna parte. Por sí mismos no pueden almacenarla, sólo pueden procesarla. Para que el ordenador pueda almacenar información, debe estar equipado de una memoria y aquí es donde intervienen las RAM y ROM.

Los ordenadores necesitan "recordar" dos tipos diferentes de información. El primer tipo está formado por las instrucciones que la CPU debe recorrer para hacer su trabajo. Estas instrucciones se agrupan en programas para hacer que el ordenador realice determinadas tareas como la ejecución de juegos o el proceso de textos. El segundo tipo de información que necesita recordar el ordenador son los datos. Estos pueden ser números o palabras utilizados por los programas –como el marcador en un juego o el texto durante el proceso de un documento.

Antes de que la CPU pueda hacer cualquier trabajo, habrá que decirla qué debe hacer (por ejemplo, multiplicar dos números), y con qué debe hacerlo (en este caso, los dos números que hemos mencionado). La CPU toma las instrucciones y los datos de la Memoria de Acceso Directo RAM (del inglés Random-Access Memory). Cuando la CPU ejecuta un programa que Vd. ha escrito o comprado para su ordenador, va a buscar las instrucciones y los datos requeridos por el programa en la RAM (a esto se le llama leer de la RAM). Si Vd. tiene un programa grabado en un disquete o en una cinta de casete debe copiarlo en la RAM para que la CPU pueda utilizarlo.

Cuando el ordenador realiza un cálculo, devuelve el resultado a la RAM (se llama **escribir** en la RAM) y desde aquí puede visualizarlo en la pantalla o grabarlo en un disco o en una cinta.

Hay una clase especial de memoria de la cual la CPU puede leer pero en la que no puede escribir. Se llama Memoria de sólo Lectura o **ROM** (del inglés Read-Only Memory). La diferencia entre las dos

memorias consiste en que cuando se le desconecta la energía al ordenador, los datos o los programas que estuviesen almacenados en la RAM se pierden (como ocurre con las calculadoras de bosillo). Si Vd. quiere volver a ejecutar un programa después de haber desenchufado el ordenador, debe volver a cargarlo en la memoria RAM desde el teclado o recuperarlo desde disco o cinta. La memoria ROM, por el contrario, conservará siempre su contenido, incluso cuando el ordenador está apagado.

Esta característica de la ROM es extremadamente útil. Cuando se enciende el ordenador, por ejemplo, inmediatamente empieza a ejecutar las instrucciones almacenadas en la ROM. La primera le dice al ordenador que verifique el estado de sus componentes. Dado que estas instrucciones están almacenadas en ROM, no es necesario cargarlas cada vez que se conecte el ordenador.

Otra utilización importante de la ROM es el almacenamiento de las instrucciones que hacen posible el funcionamiento del lenguaje MSX-BASIC. En este sentido, cada vez que el ordenador se conecta y se autocomprueba, visualiza en la pantalla un mensaje de bienvenida y un "OK" para indicar que ya se puede utilizar el MSX-BASIC.

De acuerdo con RAM y ROM, pero ¿qué significa 32K?

Bits y bytes

32K es una medida de la cantidad de memoria disponible en su ordenador. A fin de comprender lo que esto significa en la práctica, vamos a ver brevemente cómo el ordenador almacena y utiliza los datos.

El ordenador transmite y almacena la información bajo el formato de un código. Este código es necesario porque la información sólo puede ser enviada eléctricamente y no sería práctico transmitir cada elemento de información en un voltaje diferente. Así pues, el ordenador envía la información utilizando solamente dos voltajes —0 voltios y unos 5 voltios aproximadamente. Dado que sólo existen estos dos voltajes, ha habido que combinarlos en grupos para representar caracteres como, por ejemplo, las letras del alfabeto. Estos voltajes, o impulsos, se denominan **bits**. Cuando nos referimos a un bit, decimos que es un "1" para indicar 5V o una señal en "on", o que es un "0" para indicar 0V o una señal en "off".

Si, por ejemplo, se envían 3 bits (con señales de 5V o 0V) a la vez, podemos representar ocho caracteres diferentes como vemos a continuación.

Número del carácter	Primer Bit	Segundo Bit	Tercer Bit
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

A partir de esta tabla, probablemente Vd. puede ver que si dos ordenadores se ponen de acuerdo para enviarse información en fragmentos conteniendo cada uno de ellos tres impulsos, podrán comunicarse, utilizando ocho caracteres diferentes. En la práctica, no se podría hacer prácticamente nada si sólo se pudiesen enviar ocho caracteres.

En la terminología de ordenadores a cada grupo de bits se le llama "palabra". Diríamos que el código que acabamos de ver utiliza una palabra de 3 bits. Es importante que tenga Vd. bien claro que una palabra del ordenador corresponde a un único carácter de nuestro lenguaje.

Si se amplía el código a cuatro bits, el número de caracteres que el ordenador puede enviar se incrementa a 16. Puede Vd. mismo comprobarlo, escribiendo todos los diferentes códigos que se podrían enviar. Un sistema para ver cómo este bit adicional incrementa el número de caracteres consistiría en añadir un 0 delante de todos los números que hemos listado antes (con lo cual tenemos ocho números de 4 bits que empiezan todos con 0) y luego sustituir este 0 por un 1 (tenemos entonces otros ocho números de 4 bits que empiezan con 1). Todos juntos dan los 16 caracteres diferentes, cada uno con cuatro bits.

Como necesitamos enviar números y letras, signos de puntuación y caracteres especiales como *, $ y >, el ordenador requiere bastante más que cuatro bits para cada carácter. De hecho, los diseñadores y los usuarios de los ordenadores han acordado utilizar 7 bits por carácter. El código, que prácticamente utilizan todos los micro-ordenadores se llama código ASCII (American Standard Code for Information Interchange) y aparece en la figura 1.

Aunque el código ASCII sea un código de 7 bits, la información se envía en palabras de 8 bits pudiéndose utilizar este bit extra para asegurarse de que los datos se envían correctamente.

Al igual que en la transmisión de datos, el ordenador también almacena los caracteres en palabras de 8 bits y cuando nos referimos precisamente al almacenamiento se utiliza el término "byte". La diferencia

ASCII		ASCII		ASCII	
Código	Carácter	Código	Carácter	Código	Carácter
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	–	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	<
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DCI	060	>	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	ESPACIO	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035		078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

Los códigos ASCII están en decimal.

LF = Alimentación de línea (Line Feed), FF = Alimentación de página (Form Feed) y CR = Retorno de carro (Carriage Return) son ejemplos de los números de los códigos de control que van del 1 al 32.

Fig. 1.—Códigos ASCII

entre una palabra y un byte es que un byte siempre está formado por 8 bits y una palabra puede ser de cualquier longitud (como hemos demostrado con nuestras palabras de 3 bits). Cuando las palabras son de 8 bits, una palabra es lo mismo que un byte y se pueden emplear ambos términos indistintamente.

Imagínese la memoria del ordenador como una parrilla de cables. Hay ocho cables que van de arriba abajo y 16 que los atraviesan. En los puntos en los que se cruzan los cables existe la posibilidad de almacenar una carga (como si fuese un condensador pequeño) (Fig. 2).

Para almacenar el byte 10110011 (esto es un 51 en decimal; vea el carácter correspondiente en la tabla ASCII) hay que ocupar las intersecciones primera, tercera, cuarta, séptima y octava.

Como ve Vd. en este diagrama, la capacidad total de nuestra memoria es de 128 bits o 16 bytes. La memoria está estructurada en 8 × 16 bits.

Esta es una memoria increíblemente pequeña, para los ordenadores estándares corrientes, ya que en ella sólo se pueden almacenar 16 letras (se podría almacenar, por ejemplo, la palabra "PROGRAMABILIDAD.". Hace unos pocos años, 4.096 bytes eran considerados como una cantidad razonable de memoria para un micro pequeño.

En lugar de escribir 4.096 bytes, la industria del ordenador utiliza la letra "K" para representar 1.024 bytes (igual que la utilización de K para representar 1.000 como en 1 km), de manera que 4.096 se escribe como 4K. Los orígenes de este número 1.024, bastante extraño, se encuentra en el hecho de que las palabras de 8 bits suelen ser muy frecuentes en los ordenadores. Si se sigue añadiendo un bit a la longitud de la palabra, se dobla el número de palabras distintas –ya vimos esto antes cuando pasamos de una palabra de 3 bits a una de 4 bits. Si se repite esta duplicación, en un momento dado tendremos 1.024 caracteres diferentes y como toda la memoria debe ser un múltiplo o fracción de 1.024, parece lógico adoptar esta cantidad como unidad de medida en lugar de 1.000.

Como ya hemos dicho, una memoria de 4K era muy corriente en los primeros micros. Hoy en día, se escriben a menudo unos tipos de programas que requieren, para su almacenamiento y ejecución, mucha más memoria por lo que ahora son normales las memorias de 8K y 16K. En los grandes micro-ordenadores del mundo de los negocios es más frecuente encontrar memorias de 64K y 128K (observe que 128K es el doble de 64K). A menudo se amplían a 256K, 512K o incluso hasta 1.024K. En este punto, la letra K se sustituye por la letra M para indicar Megabytes, donde 1 Megabyte equivale a 1.024 Kilobytes. En la figura 3 se sumariza la nomenclatura que hemos comentado hasta aquí.

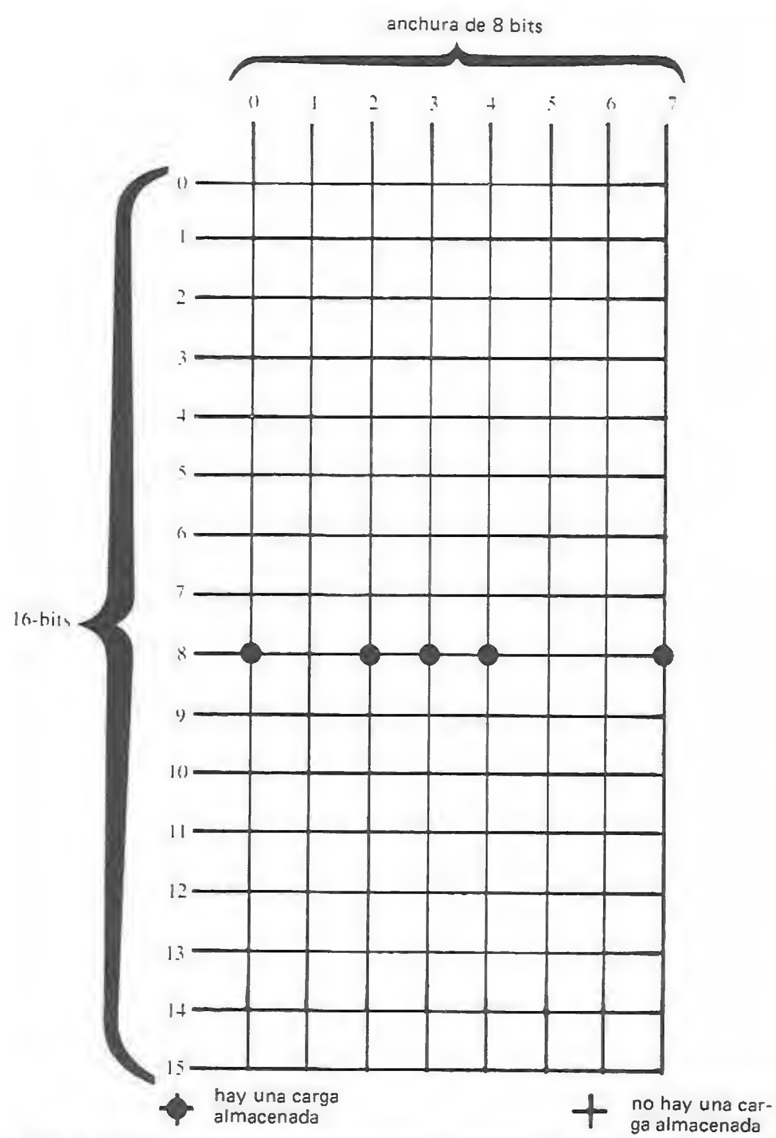


Fig. 2. -*Representación en diagrama de las celdas de memoria. Los bits en "ON" ó 1s están representados por discos negros en las intersecciones de la malla. Los bytes se referencian por su "posición" dentro de la malla. En la posición 8 de memoria se ha almacenado la palabra binaria 10111001.*

1 bit = señal eléctrica de 0V ó 5V
1 palabra = un número de bits
(el número depende del ordenador
pero suele ser 8 ó 16).
1 byte = 8 bits
1 Kilobyte (K) = 1.024 bytes
1 Megabyte (M) = 1.024 kilobytes

Fig. 3.—Sumarización de la nomenclatura

Su ordenador MSX está basado en un microprocesador que utiliza palabras de 8 bits y trae una memoria ROM de 32K y una RAM comprendida entre 32 y 64K.

Un punto importante a considerar es que cuanto mayor y más complicado sea el programa que Vd. desea ejecutar en su ordenador, más memoria necesita el ordenador para almacenarlo. Todos los ordenadores MSX tienen la capacidad de añadir más RAM o ROM y esto es algo que debe Vd. tener en mente cuando empiece a escribir programas grandes. También merece la pena considerar el hecho de que aunque su ordenador incluye un mínimo de 32K de RAM, 16K se dedican al almacenamiento de la información utilizada por la pantalla, de manera que sólo le quedan 16 (si su máquina trae únicamente 32K) para Vd. Dicho esto, añadiremos que todos los programas de este libro caben en este tamaño, lo cual le dará una idea de todo lo que puede hacerse, incluso con una cantidad limitada de memoria.

Cartuchos de ROM

Si bien la vida de la industria del ordenador casero es muy corta, la forma tradicional de pasar el tráfico de la información a y desde el ordenador ha sido la utilización de una grabadora de casetes. Nos detendremos en este punto próximamente, pero ha sido un método que ha arrastrado muchos problemas. En comparación, el cartucho ROM es todo simplicidad.

Cuando se incluye un cartucho ROM en el ordenador, está realmente sustituyendo a la ROM que viene de fábrica. En lugar de acceder como antes al MSX-BASIC, le aparecerá en la pantalla un juego, un programa de tratamiento casero o cualquier cosa para lo que Vd. haya comprado el cartucho ROM. Esto no sólo proporciona un método sencillo para cargar programas en el ordenador, también significa que el tamaño de su programa ya no está limitado por la cantidad de RAM de su ordenador. No hay ninguna razón por la que no pueda Vd. cargar un cartucho de 64K ROM aunque en su ordenador sólo haya 32K de RAM.

El otro beneficio importante es que Vd. ya no tiene que esperar tiempos interminables para cargar un programa desde el casete, a veces, solamente para comprobar que algo ha ido mal y que hay que empezar otra vez.

El teclado

Para completar esta visión del hardware, nos fijaremos, por último en el teclado. Hay muchas y buenas razones para incluir el teclado en la sección de periféricos, como verá cuando la lea. La razón por la que hemos incluido el teclado aquí es que incluye una característica de muchos de los mejores ordenadores caseros —teclas de función programables. Como Vd. puede ver cuando conecta su ordenador, hay cinco teclas que, utilizadas conjuntamente con la tecla SHIFT, pueden realizar diez funciones especiales. Si está Vd. interesado en cambiar la funcionalidad de estas teclas, eche un vistazo al mandato KEY del siguiente capítulo.

Una vez que hemos analizado lo que su ordenador puede hacer y cómo obtiene sus resultados, explicaremos ahora cómo se comunica con el mundo exterior a través de sus periféricos.

Periféricos

Un periférico es, algo bastante sencillo, cualquier cosa que Vd. pueda conectar a un ordenador. Con una definición como ésta, el término cubre un rango enorme de dispositivos incluyendo aquellos que le permiten pasar información a y desde el ordenador, almacenar información o realizar tareas especiales. Veremos estos diferentes tipos en orden.

Entrada y Salida

El sistema más obvio de suministrar información al ordenador es vía el teclado por lo que, ya puede suponer nuestra resistencia a su inclusión en la sección del hardware. Como permite la introducción de información se le denomina dispositivo de entrada (input). La especificación del MSX no incluye un diseño de teclado estándar, sin embargo, todos los ordenadores MSX tienen el mismo número de teclas de función que ya hemos descrito en la sección anterior y aunque a primera vista parezca que sólo permiten el ahorro de algo de tecleo, se les puede asignar

una gran variedad de funciones como se demostrará en los capítulos posteriores.

Antes de estudiar otros dispositivos de entrada, veamos brevemente el dispositivo estándar de *salida* –la pantalla de TV. Casi todos los ordenadores caseros actualmente disponibles utilizan el televisor como el medio principal para visualizar información porque, igual que un grabador de casetes, cualquiera que compre un ordenador casero seguro que ya tiene uno.

Dada la presión creciente sobre los televisores caseros de fuentes tales como vídeos, teletexto y otras, los monitores en color proporcionan una solución barata para los usuarios que se ven relegados de su sofá. Los monitores no sólo son más baratos que un segundo televisor, la calidad de la pantalla y del color son superiores y en ellos, los gráficos del MSX son mucho más impresionantes.

Para aquellos que utilicen muy poco las facilidades del color del MSX un sencillo portátil blanco y negro (o un monitor) proporciona la más barata de todas las soluciones. Incluso sin color, los gráficos del MSX siguen siendo sorprendentes (como comprobamos mientras escribimos este libro).

Aunque brevemente, hemos descrito los dispositivos estándares de entrada y salida, el resto en realidad son dispositivos alternativos que se utilizan para aplicaciones específicas. Alternativos del teclado, por ejemplo, son los mandos de juego (que dan una respuesta mucho más rápida que el teclado para jugar con el ordenador), los lápices luminosos (su aplicación es obvia conjuntamente con un paquete de dibujos), y los teclados de bolsillo (utilizados sobre todo cuando el ordenador visualiza en la pantalla un rango de opciones que se corresponden con las teclas del teclado de bolsillo). Los teclados de bolsillo, en esencia, proporcionan una ampliación de las teclas de función. Para cuando Vd. lea este libro, seguramente existirán otros dispositivos alternativos que ni siquiera conocemos, cada uno construido para una función específica y con numerosos anuncios alabando sus virtudes en las tiendas populares de ordenadores.

Impresoras

Volviendo a los dispositivos alternativos de salida, los más elementales son las impresoras, le permiten guardar una "copia dura" (hard copy = una copia en papel) de su trabajo. Los tres tipos más importantes se describen a continuación.

Las impresoras matriciales son las más baratas en la actualidad. Los caracteres se configuran mediante una matriz de puntos que normal-

mente suele ser de siete puntos en la vertical por cinco en la horizontal aunque las matrices de 9 X 7 también son muy corrientes. Los puntos se marcan en el papel bien sea por una serie de puntas que se presionan contra una cinta estándar de impresora, o bien por puntas calientes que se presionan contra un papel sensible al calor. Pueden imprimir caracteres estándares y, en algunos casos, un rango limitado de caracteres gráficos.

Las impresoras **de margarita**, por el contrario, incorporan un mecanismo más complicado y su modo de operar es comparable con el de las máquinas de escribir de bola. Cada letra o símbolo del juego de caracteres se encuentra en un "pétalo" de la margarita. La margarita gira a alta velocidad y un martillo golpea el pétalo de un carácter determinado contra la cinta, sobre el papel. Su puede obtener una alta calidad de impresión (calidad de carta) porque las margaritas se fabrican de plástico o de metal. Además las margaritas se cambian rápida y fácilmente permitiendo así la impresión con distintos tipos de letras, en distintos idiomas, etc. Las impresoras de margarita son generalmente menos flexibles que las impresoras matriciales: por ejemplo no pueden imprimir gráficos y en determinados casos no se puede hacer una copia de la pantalla. También son más lentas (debido a la forma en que trabajan) y más caras. Su principal ventaja sobre las impresoras matriciales es que producen una calidad de impresión muy superior, aunque esto se consigue a costa de ser mucho más lentas.

Por último, están los **trazadores de gráficos** o **plotters** utilizados para obtener una copia dura de cualquier gráfico generado por el sistema. Funcionan utilizando una pluma, o plumas de diferentes colores en el caso de plotters de color, para dibujar con absoluta precisión la imagen que aparece en la pantalla. Los plotters son máquinas mucho más complejas que las impresoras y su micro MSX, en su estado original, no las puede manejar. Requieren un elemento especial de software que se llama conductor de dispositivo (device driver) que traduce lo que aparece en la pantalla a mandatos que hacen que el plotter dibuje. Los conductores de dispositivos requieren un **sistema operativo (MXS-DOS)**, y **discos flexibles** (floppy): de todo esto hablaremos más adelante. Por ahora sólo diremos que esto hace que estos dispositivos sean mucho más caros que otros dispositivos de salida y sólo son aplicables a aquellos usuarios que por su trabajo deben obtener gráficos complejos, como por ejemplo planos de ingeniería, diseños de tarjetas de circuitos impresos, etc.

Es casi seguro, que el MSX estándar va a producir toda una nueva gama de impresoras y plotters compatibles con el MSX que seguramente dejarán los precios tambaleándose y al alcance de la mayoría de los usuarios.

Dispositivos de almacenamiento

Y de los dispositivos que le permiten pasar información a/desde el ordenador, pasamos a los que le permiten almacenar información. Ya hemos hablado de la RAM y la ROM del ordenador. El problema que tiene la RAM, como ya hemos visto, es que cuando Vd. ha pasado varias horas escribiendo un programa desde el teclado y quiere parar y apagar la máquina, el programa se pierde. Lo que se necesita es una memoria, o un dispositivo de almacenamiento que sea capaz de guardar los datos aunque se desconecte la energía.

Esencialmente hay dos tipos de dispositivos de almacenamiento para los micros caseros –grabadores de cinta y discos flexibles o disquetes. Ambos tienen ventajas e inconvenientes. Ya hemos visto algunos de los problemas con los que se puede encontrar en la utilización de casetes para el almacenamiento de datos, pero realmente proporcionan un método muy barato, aunque lento, de almacenamiento de la información. Los disquetes por el contrario, pueden albergar cantidades de información mucho mayores (de 100K hasta 400K u 800K en un sólo disco). Son rápidos pero relativamente caros. Dicho esto, los disquetes le serán esenciales si Vd. va a utilizar el ordenador para algún negocio serio, en lugar de tenerlo como hobby.

Robots y sintetizadores de voz

El último tipo de periféricos que vamos a ver, y que esperamos que cada vez sean más numerosos, son aquéllos diseñados para realizar funciones específicas. Podemos incluir en esta categoría a periféricos como los mandos de juego, lápices luminosos y teclados de bolsillo. Además están los dispositivos como los sintetizadores de voz, los dispositivos de reconocimiento de la palabra hablada, robots, y cualesquiera otros que por el momento puedan sonar a ciencia-ficción pero que se convertirán, y en breves fechas, en la realidad cotidiana. Como ya mencionamos en el prólogo del libro, su ordenador MSX es, en principio, capaz de hablar con cualquier dispositivo que Vd. pueda imaginar. Todo lo que se necesita es que alguien aporte dispositivos domésticos sencillos capaces de ser conectados, bien directamente, bien a través de una interfase, a un ordenador MSX.

Con esta simpática observación, y con objeto de prepararle para el siguiente capítulo, entramos en el tema del software del ordenador.

Software

Ya le hemos explicado los fundamentos del hardware del ordenador MSX, y de los dispositivos que se le pueden conectar para pasar información a/desde el ordenador, pero ¿cómo hacer que toda esta tecnología haga las cosas que Vd. quiere que haga? La respuesta es que Vd. le dice qué es lo que debe hacer, mediante el uso de programas o "**software**".

Como ya dijimos en la sección del hardware, el lenguaje entendido por el procesador central es el código máquina. Resulta bastante improbable que haya escrito Vd. algún programa en este lenguaje porque es extremadamente difícil de comprender y su aprendizaje lleva mucho tiempo. Obviamente, sería mucho más fácil si Vd. simplemente le dijese al ordenador en lenguaje natural qué es lo que Vd. quiere que haga. Desgraciadamente todos los lenguajes naturales aportan una gran ambigüedad haciendo que esto sea imposible. Sin embargo, Vd. verá lo parecido que es el lenguaje de programación BASIC a la programación en inglés.

El BASIC de Microsoft es la implantación estándar industrial del lenguaje, tanto en el aspecto de su sofisticación como de su popularidad. Actualmente está instalado en más de 2 millones de micro-ordenadores por todo el mundo. El MSX-BASIC ha sido desarrollado a partir del BASIC de Microsoft e incorpora muchos mandatos especiales para aprovechar todas las ventajas del hardware del MSX. Lo que el MSX-BASIC hace, esencialmente, es coger unas instrucciones escritas en un lenguaje comprensible para la gente y traducirlas a un lenguaje comprensible para la máquina.

En los próximos capítulos veremos con más detalle, cuáles son las instrucciones del MSX-BASIC y qué es lo que hacen. Cualquier lenguaje de ordenador se compone de **mandatos**, o **comandos**, **funciones** y **sentencias** que pueden ser utilizados por sí solos o en combinación con los demás en cuyo caso constituyen un programa.

La colocación de todas las instrucciones requeridas del MSX-BASIC juntas para formar un programa precisa una aproximación organizada a las soluciones del problema. Como ayuda, durante esta aproximación, los programadores de ordenadores utilizan lo que se llama un **organigrama** (flowchart) para seccionar el problema en sus partes constituyentes.

Cuando un problema queda definido de esta forma, es muy sencillo traducir las distintas partes a instrucciones comprensibles para el MSX-BASIC. Cuando todas estas instrucciones se agrupan tenemos un programa. Todo lo que se necesita es un poco de imaginación para ver cómo se puede trasladar a un organigrama el problema que Vd. quiere resolver.

A estas alturas, resulta conveniente ver todos y cada uno de los mandatos, funciones y sentencias que el MSX-BASIC proporciona al programador.

Mandatos

Los mandatos le dicen al ordenador que haga algo como RUN (ejecutar) o LIST (listar). Los mandatos se ejecutan nada más teclearlos y no necesitan formar parte de su programa.

Sentencias

Las sentencias son las instrucciones numeradas que configuran un programa. LET X = 5 es un ejemplo de sentencia. Asigna a X el valor 5.

Funciones

El MSX-BASIC ofrece un número de funciones matemáticas como el seno, el coseno y la tangente. Las funciones van siempre seguidas de un número, una variable o una expresión entre corchetes. Un ejemplo de función es SIN (la función seno). PRINT SIN (PI/2) dará el seno de 90° en radianes ¡Seguramente lo recuerda de sus días de colegial!

Así es, en esencia, como funciona el lenguaje. Hemos cubierto muchos temas (con su jerga correspondiente) en esta sección así que quizás sea conveniente terminar con un programa que sea ejemplo de programación "típica".

Principios de programación

Ya dijimos al principio que íbamos a intentar evitar los problemas con las cuentas bancarias y los intereses compuestos: son temas muy próximos al corazón de cada uno, y además nos proporcionan un método adecuado para la demostración de las ideas de los que hemos estado hablando. No tema, los ejemplos restantes que le vamos a presentar serán mucho más interesantes.

He aquí el problema. Yo he ingresado 10.000 ptas. en la cuenta de una sociedad constructora. Asumiendo que las ganancias sean de un 10% de interés anual y que el tipo de interés se mantiene constante ¿qué dinero tendré al cabo de 10 años?

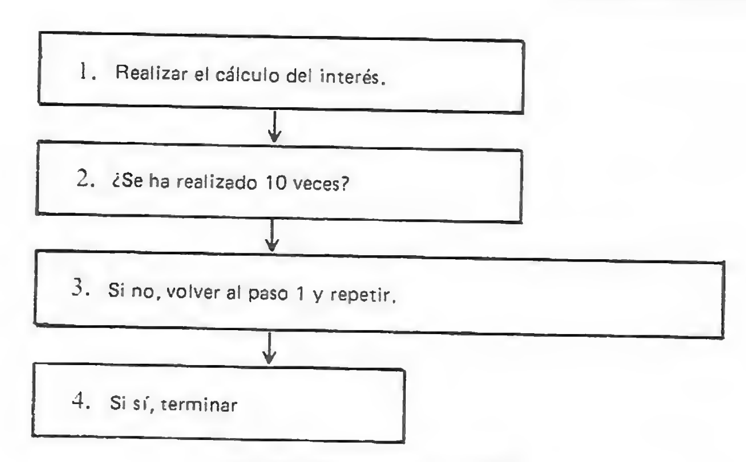


Fig. 4.—Organigrama sencillo

La primera fase del análisis del problema es su desglose en componentes más pequeños.

La tarea comprende tres etapas principales. En primer lugar hay que decirle al ordenador el valor inicial del dinero depositado en la cuenta. A continuación hay que calcular los intereses del primer año. Y repetir esto 10 veces. Representando estos pasos en un organigrama, se puede llegar a uno similar al de la figura 4.

Estos pasos también aparecen en la figura 5 donde vemos cómo utiliza el BASIC las variables y cómo el ordenador puede realizar determinadas funciones como "repetir diez veces".

Una variable es una "caja" que el ordenador utiliza para almacenar la información que se va a modificar. En el ejemplo anterior, sabemos que la cantidad de dinero de la cuenta, y el número de años en las que se hace la inversión deben modificarse para llegar a la solución del problema. Las variables "Año", que crecerá de 0 a 10, y "Dinero" que hará aumentar la cantidad inicial de 10.000 a razón del interés compuesto del 10% anual, se declaran al comienzo del organigrama. La variable Año se compueba cada vez que se completa un bucle para ver si ha alcanzado el valor 10. Si no lo ha alcanzado se ejecuta el bucle otra vez. En caso contrario, el valor que hay en Dinero corresponde a la cantidad que habrá en la cuenta al cabo de 10 años, y el programa, que prácticamente es esto, se para.

Si Vd. entiende todo esto, no tendrá ningún problema para comprender los siguientes capítulos. Lo que vamos a hacer es presentarle las

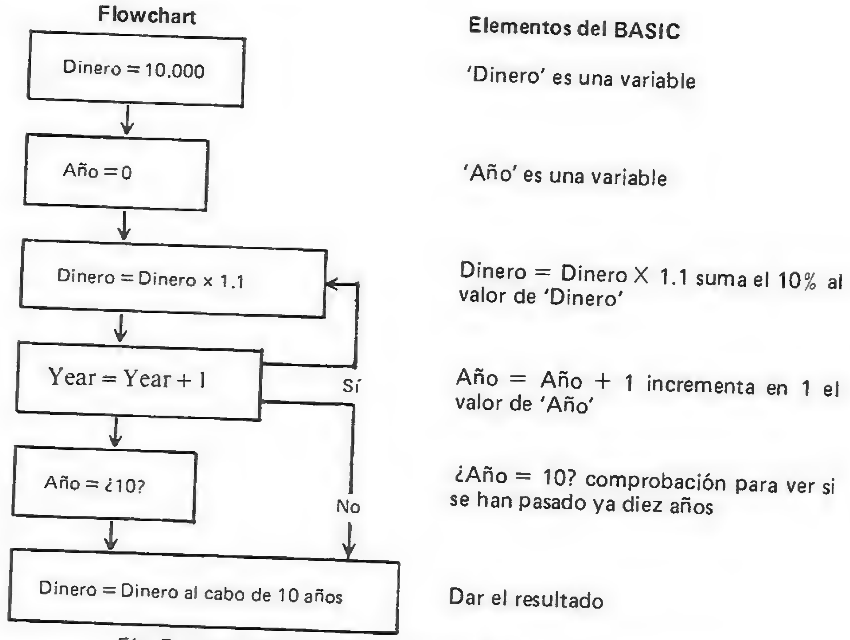


Fig. 5.—Organigrama del programa de la sociedad constructora

instrucciones que están disponibles y lo que hacen, para después mostrarle cómo puede conectarlas para ir formando programas cuya complejidad vaya aumentando progresivamente.

De cualquier manera, esperamos que esta introducción a los ordenadores en general, y al MSX en particular, le resulte provechosa. Ahora Vd. ha llegado al final de la parte preliminar del libro y ya es hora de alguna participación de los lectores, así que conecte su ordenador y practique. ¡Es hora de que probemos algunos mandatos!

Cómo programar en MSX-BASIC

Mandatos interpretados

Cuando Vd. enciende su ordenador MSX, aparece en la pantalla el mensaje "OK" para decirle que el ordenador está preparado para recibir y procesar instrucciones del BASIC. Esto se puede conseguir de dos maneras. La primera consiste en introducir las instrucciones directamente, en cuyo caso el ordenador interpreta y actúa sobre ellas inmediatamente (esto se conoce como modalidad directa). En la segunda, las instrucciones van precedidas por números de líneas, en cuyo caso son tratadas como si fuesen un programa y ejecutadas en el orden en el que aparecen en el programa. Un programa se ejecuta en modalidad indirecta y sólo empieza cuando se teclea el mandato RUN directamente. Desde luego, la forma más sencilla de ver qué es un mandato y qué es lo que hace, consiste en teclearlo y ver cómo lo interpreta el ordenador. Como ejemplo teclee lo siguiente:

```
PLAY "Ø4CDEFGABØ5C"
```

y pulse la tecla RETURN. PLAY le dice al ordenador que debe ejecutar alguna música. "Ø4CDEFGABØ5C" le dice al ordenador exactamente qué es lo que debe tocar. Como Vd. puede oir, el resultado de este tecleo es que el ordenador ejecuta una escala completa. Más adelante, ya verá cómo se consigue esto.

La mayoría de las instrucciones del MSX-BASIC, también de otros BASICs, se componen del nombre de la instrucción y una expresión en la que se define cómo hay que utilizarla. Cuando se está utilizando una instrucción es vital que la sintaxis de las notaciones que van a continuación sea correcta y ésta es una buena razón para estudiar primero la modalidad indirecta. Dicho esto añadiremos que algunas de las instrucciones que le vamos a presentar en esta sección irán incluidas en un programa para que tengan algún significado. Cuando llegue este caso, haremos que el programa sea lo más corto posible aunque se demuestre totalmente cómo se puede utilizar la instrucción. Cuando tenga que ejecutar un programa para ver cómo funciona la instrucción, por favor, haga lo siguiente:

1. Teclee el programa exactamente como aparece impreso en el libro.
2. Escriba el mandato RUN sin que le preceda ningún número de línea (modalidad directa) o pulse la tecla de función 5 de su teclado.
3. La mayoría de los programas que Vd. escriba, por sí sólo, terminarán con la reaparición del mensaje de diálogo del MSX-BASIC "OK" en la pantalla. Para los programas que siguen. Vd. debe pulsar las teclas CONTROL y STOP simultáneamente para detener el programa y volver al mensaje OK. Si va a modificar el programa, puede teclear LIST para listarlo, o NEW para borrarlo y volver a introducir otro nuevo.

Una de las características del MSX-BASIC es que Vd. puede "**editar**" sentencias en las pantallas. En el ejemplo anterior, en el que utilizamos PLAY, Vd. puede subir el cursor y cambiar sólo algunas de las letras. Pulsando la tecla ENTER o la tecla del retorno de carro se produce la introducción de las líneas modificadas, exactamente igual que si Vd. la hubiese tecleado por completo en una línea vacía.

Esto es válido para cualquier línea de la pantalla. Si Vd. escribe una línea y se mantiene todavía visible en la pantalla, puede volver a ella, editarla y salvarla de nuevo pulsando la tecla ENTER. Si se olvida de pulsar la tecla ENTER, la línea permanece tal y como estaba antes de que Vd. hiciera los cambios.

Esta facilidad es muy útil en los programas MSX-BASIC, en los que puede haber líneas largas y complicadas conteniendo algún error. Se puede listar dicha línea en la pantalla y corregirla sin tener que volver a escribir toda la línea. Le sugerimos que practique y coja experiencia con los programas ejemplos de este libro. Una vez que haya escrito nuestros programas, debe utilizar la facilidad de edición para experimentar distintos tipos de modificación que se le puedan ocurrir.

Todas las características del Editor del MSX-BASIC, así es como se llama, están relacionadas en el Apéndice C. Es conveniente que, antes de continuar, se familiarice con la forma de trabajar con el Editor.

Si todo va conforme es debido, cuando Vd. finalice esta sección y la siguiente sobre gráficos, habrá conseguido dos cosas. Primero y principalmente, habrá comprendido las instrucciones del BASIC más corrientemente utilizadas –lo que hacen y cómo lo hacen. En segundo lugar, a partir de los programas ejemplos que hemos incluido en esta sección se habrá formado una idea sobre lo que es un programa BASIC y estará preparado para la última sección del capítulo en la que se describen las formas de escribir programas, más complejos y por lo tanto más útiles.

Como inciso, podría ser apropiado introducir aquí la notación que vamos a utilizar para describir cada instrucción. Como verá Vd. en el mandato AUTO (más abajo), la sintaxis incluye los símbolos [ ] y < > y las últimas sentencias los símbolos ( ). Tienen los siguientes significados:

< > Cualquier cosa que aparezca dentro de estos signos mayor y menor debe ser utilizado con la instrucción, en caso contrario el ordenador no lo reconoce.

[ ] Lo que va entre corchetes es opcional. Depende de Vd. su inclusión. Si no lo incluye, el ordenador asume que Vd. se conforma con la condición de omisión o valor por defecto (por ejemplo en el mandato AUTO si Vd. no especifica la línea inicial o el incremento, la numeración empezará en la línea 10 con un incremento de 10).

( ) Son los únicos símbolos que debe Vd. teclear ya que forman parte de la sintaxis. Ya verá cómo hay que utilizarlos exactamente cuando lleguemos a una instrucción que haga uso de ellos.

Bien, empecemos a ver con detalle los mandatos. ¡Buena suerte!

AUTO [<línea inicial>] [, <incremento>]

Todas las líneas de un programa tienen que estar numeradas. El ordenador ejecuta las instrucciones contenidas en las líneas, en orden numérico. El mandato AUTO numera automáticamente las líneas según se van introduciendo al ordenador, ahorrándole a Vd. el tiempo que se precisa para esta tarea. AUTO se utiliza solamente, cuando se está tecleando un programa.

```
AUTO 100, 10
```

numeraría las líneas de 10 en 10 y empezando en 100;

```
AUTO 100, 20
```

daría un incremento de 20 y así sucesivamente.

Si Vd. utiliza AUTO en un programa parcialmente escrito, si el mandato genera un número de línea que ya existe, aparecerá un asterisco al final de la línea para avisarle de que esta nueva línea sustituirá a la existente. Para desactivar el mandato, pulse las teclas CONTROL y STOP simultáneamente.

BEEP

Cuando Vd. teclea BEEP se genera un sonido. Es el más elemental de los mandatos que generan sonidos. Para otros, algo más divertidos, vea el mandato SOUND o pase al capítulo musical.

CLEAR [<cadena de espacios>] [,<dirección más alta>]

El mandato CLEAR borra todas las variables de la memoria del ordenador sin que se borre el programa actualmente almacenado en RAM. Después de la ejecución de un programa se mantiene asignadas en el RAM las variables que ha utilizado. Para simplificar, lo que hace es borrar de la memoria todo aquello que Vd. ya no va a necesitar.

CLS

CLS simplemente borra todo lo que hubiese en la pantalla.

COLOR [<color de primer plano>] [,<color de fondo>] [,<color de borde>]

El mandato COLOR se utiliza para establecer los colores de la pantalla. Como se deduce de su sintaxis, es bastante flexible permitiendo controlar el color del primer plano, el de fondo y el de borde mediante un comando de una sola línea. Cuando se conecta el ordenador, es el mandato COLOR el que establece los colores que ve Vd. en la pantalla. El ordenador tiene un valor "por defecto" u "omisión" para COLOR que es 15,5,4 correspondiendo estos números al blanco, azul oscuro y azul claro respectivamente. El defecto es lo que toma una instrucción si no se añaden comentarios, o texto.

Nota: Es importante hacer aquí una observación, ya que de acuerdo con las características de su televisor, es posible que Vd. no pueda ver el color de borde durante todo el tiempo. Sin embargo aparecerá cuando se esté ejecutando un programa (suponiendo que lo haya especificado, claro está). Por razones de sencillez, hemos intentado evitar la utilización de programas en esta sección. Pero en este caso es inevitable un programita corto únicamente para demostrar que existen estas tres áreas en la pantalla. Escriba:

```
10 SCREEN 2
20 COLOR 14,2,9:CLS
```

```
30 CIRCLE (125,100),50
40 GOTO 40
```

Lo que debe aparecer es un círculo gris sobre un fondo verde y rodeado de un borde rosa.

Vd. puede utilizar el mandato COLOR para establecer en la pantalla los colores que le resulten más cómodos. Si escribe, por ejemplo, COLOR 12, 15, los caracteres se escribirán en verde sobre un fondo blanco y es conveniente que Vd. experimente y haga distintas pruebas hasta que encuentre los que le parezcan más convenientes. A continuación damos la relación de los distintos colores y sus números correspondientes.

Número	Color
0	Transparente
1	Negro
2	Verde normal
3	Verde claro
4	Azul oscuro
5	Azul claro
6	Rojo oscuro
7	Azul grisáceo
8	Rojo normal
9	Rojo claro
10	Amarillo fuerte
11	Amarillo claro
12	Verde oscuro
13	Magenta
14	Gris
15	Blanco

CONT

El comando CONTinuar se utiliza en un programa BASIC, o en modalidad directa, para indicar al ordenador que debe continuar la ejecución de un programa previamente parado, por ejemplo después de que el programa haya ejecutado un mandato STOP o BREAK.

DELETE [<número de línea inicial>] [–<número de línea final>]

El mandato DELETE elimina, de un programa, las líneas que no se quieren mantener. Todo lo que hay que especificar son las líneas primera y última que hay que borrar. Por ejemplo, DELETE 20-40 borrará

del programa desde las líneas 20 hasta la 40 ambas inclusive; DELETE 20 borrará solamente la línea 20 y así sucesivamente. Una forma más sencilla de borrar líneas (de una en una) consiste en teclear sólo el número de línea seguido de ENTER. Entonces el ordenador almacena una línea en blanco en lugar de la que existía previamente.

KEY <número de tecla> , "<mandato>"

El ordenador MSX incluye 5 teclas programables que, junto con la tecla SHIFT, permiten almacenar 10 funciones pre-programadas. Para obtener una relación de las funciones asignadas por defecto, escriba KEY LIST. Se puede cambiar cualquiera de las funciones pre-programadas utilizando el comando KEY. Teclee, por ejemplo:

```
KEY 3, "COLOR 12,15"
```

Si se pulsa ahora la tecla de función 3, se cambia la pantalla para visualizar el texto verde sobre fondo blanco. Se puede programar cualquier tecla de función para que produzca cualquier instrucción de programación con lo que se consigue un ahorro considerable de tiempo y esfuerzo cuando se teclean mandatos utilizados con mucha frecuencia. Las etiquetas de las teclas se visualizan en pantalla (¡como Vd. puede ver!); se puede hacer desaparecer esta visualización utilizando KEY OFF y aparece de nuevo con KEY ON. Para hacer que el mandato se ejecute inmediatamente es preciso incorporarle un retorno de carro (o ENTER). Esto se consigue añadiendo "CHR$(13)" a la designación de la tecla, por ejemplo:

```
KEY 1, "TRON" + CHR$(13)
```

desencadena el mandato de traza (TRON) al pulsar la tecla 1.

LIST [<número de línea>] – [<número de línea>]

El mandato LIST se puede utilizar de distintas formas para listar un programa. Si sólo se teclea LIST, se visualiza todo el programa que en ese momento, está cargado en la memoria del ordenador. Opcionalmente se pueden añadir números de línea al mandato, por ejemplo:

LIST 20-40 lista desde la línea 20 a la 40, ambas inclusive, del programa actual

LIST 20 lista la línea 20 solamente

LIST 20– lista todas las líneas desde la 20 en adelante

LIST –40 lista todas las líneas hasta la 40 inclusive

Se puede detener el listado pulsando la tecla STOP. Pulsando STOP otra vez, se reanuda el listado. Pulsando las teclas CONTROL y STOP simultáneamente se cancela el listado.

LLIST [<número de línea>] – [<número de línea>]

LLIST funciona exactamente igual que LIST, la única diferencia que existe entre los dos mandatos consiste en que en lugar de visualizar el programa en la pantalla, LLIST lo envía a la impresora –suponiendo siempre que hay una conectada al ordenador.

MAXFILES = <expresión>

Como ya hemos mencionado, siempre que se ejecuta un programa BASIC, el ordenador establece el número de ficheros que pueden controlar en la memoria para gestionar el trabajo que tiene que hacer en cada momento. El mandato MAXFILES le permite a Vd. limitar el número de ficheros que el ordenador puede abrir durante la ejecución del programa, así MAXFILES = 10 permite la apertura de 10 ficheros.

El número de ficheros puede variar de 0 a 15. Cuando se utiliza MAXFILE = 0, sólo se pueden realizar las funciones SAVE y LOAD.

MOTOR [ON] [OFF]

El mandato MOTOR se utiliza cuando su ordenador MSX tiene conectada una grabadora de casetes dedicada y sirve para activar o dasactivar el motor. Esto es muy útil, por ejemplo, cuando se está escribiendo un programa largo y sólo se desea almacenar algunas partes. Se pueden insertar los mandatos MOTOR ON/OFF en los puntos apropiados del programa.

NEW

Siempre que acaba de trabajar con un programa, puede descargarlo de la memoria de dos formas. La primera, digamos que no es muy ortodoxa, consiste en apagar el ordenador y volver a conectarlo. La segunda

consiste en teclear NEW, con lo cual se borra inmediatamente del ordenador el programa cargado en este momento, así como las variables o ficheros asociados, permitiendo así la carga o el tecleo de un nuevo programa.

RENUM [<nuevo número>] [, <antiguo número>] [, <incremento>]

El mandato RENUMerar se utiliza para renumerar las líneas de un programa BASIC, por ejemplo cuando se han añadido nuevas líneas en una zona del programa y ya no se pueden intercalar más sentencias. Se puede utilizar el mandato de varias formas.

Si sólo se escribe RENUM, se renumerarán todas las líneas del programa con múltiplos de 10, empezando (naturalmente) con la línea 10. Así, un programa que tiene las líneas 10, 13, 15, 20 y 30 quedará renumerado como 10, 20, 30, 40 y 50. Esto se modifica si se utilizan como parte de la sintaxis, un número nuevo y uno antiguo. Por ejemplo, si a un programa que tiene las líneas 10, 20, 30, 40 y 50 se le hace RENUM 15, 10 se obtiene como resultado 15, 25, 35, 45 y 55 –la línea 10 se renumera como 15 y el resto de líneas que siguen a la 15, con incrementos de 10. Si el ordenador encuentra un número de línea que ya existe, se crea un nuevo número. Por último, también se puede cambiar el incremento de líneas de manera que, por ejemplo, RENUM, 20 cambiará las líneas 10, 20, 30 a 10, 30, 50. Cualquier número de línea incluido en una línea de mandato, por ejemplo, en una sentencia GOTO, quedará modificado también por esta renumeración; de esta forma se mantiene la estructura del programa.

RUN [<número de línea>]

El mandato RUN indica al ordenador que se debe ejecutar un programa, una vez tecleado. Si sólo se escribe RUN, el ordenador ejecutará todo el programa –es decir, todas las líneas que se han tecleado. Si se especifica un número de línea, el ordenador ejecuta el programa desde ese punto en adelante. Por ejemplo, con un programa que contenga las líneas 10, 20, 30, 40 y 50, RUN hace que se ejecuten todas las líneas mientras que RUN 30 hace que se ejecuten solamente las líneas 30, 40 y 50.

SOUND

El comando SOUND se puede utilizar para hacer que la pastilla (Chip) de sonido de un ordenador haga cosas verdaderamente maravillo-

sas. Seguramente quedará sorprendido por la enorme variedad de sonidos que puede producir su ordenador, que veremos en el capítulo sobre la música, pero por ahora, debe Vd. ejecutar el siguiente programa:

```
10 SOUND 8,5
20 FOR I=1 TO 255 STEP 2
30 SOUND 0,I
40 NEXT I
50 GOTO 20
```

(Puede ser muy útil si se incendia su casa).

TRON

Es difícil, por no decir que prácticamente imposible, conseguir que el programa que uno está escribiendo, realice exactamente lo que debe ejecutar, cuando se teclea por primera vez. Incluso, excepto para los programas más pequeños, hay que dedicar una cantidad considerable de tiempo en el "debugging" es decir, en la detección y corrección de los errores que tiene el programa. Para facilitar este procedimiento el MSX–BASIC incluye el mandato TRACE que se activa tecleando TRON (TRACE ON). TRACE sigue el programa, visualizando los números de línea del programa según se van ejecutando. Así Vd. puede ver cómo está funcionando el programa durante su ejecución. TROFF es el opuesto de TRON ya que desactiva el TRACE. Raramente se utilizan estos mandatos en un programa. Normalmente se utilizan en modalidad directa antes y después de que se ejecute el programa.

TROFF

Vea el apartado anterior.

WIDTH <anchura de pantalla en modalidad de texto>

El mandato WIDTH establece la longitud de línea, es decir, el número de caracteres por línea que se pueden visualizar en pantalla en modalidad de texto. Si se ha seleccionado la opción 40 X 24 SCREEN 0, el valor dado para la anchura de la pantalla puede llegar a ser hasta 40 inclusive. Si se selecciona 32 X 24 SCREEN 1 la anchura puede llegar a ser hasta 32.

Con el mandato WIDTH concluimos este sumario de los comandos BASIC elementales. Ahora podemos ver brevemente los mandatos especiales de gráficos incluidos en MSX-BASIC (encontrará una descripción detallada en el capítulo dedicado a este tema), antes de estudiar cómo se pueden combinar estos comandos incluyéndolos en programas que hacen cosas muy útiles.

Mandatos de gráficos

Hasta ahora sólo le hemos mostrado parte de las posibilidades de su ordenador; su capacidad para visualizar 23 líneas de texto de 40 caracteres cada una, además de la línea para las teclas de función.

En términos de ordenador, decimos que esta pantalla tiene una resolución de 920 (23 × 40). Se puede producir una imagen utilizando caracteres ASCII normales. No se podrán representar imágenes con mucho detalle aunque se pueden dibujar dibujos sencillos y diagramas de barras, por ejemplo.

Algunos caracteres, llamados **caracteres gráficos** se han incorporado al MSX-BASIC para que Vd. pueda realizar dibujos más elaborados que los que se pueden conseguir utilizando sólo las letras del alfabeto o los signos de puntuación. Su microordenador MSX puede que tenga, o no, estos caracteres etiquetados en el teclado: un buen camino para comprobar cuáles son los caracteres disponibles consiste en ejecutar el programa que damos a continuación. Lo que este programa hace es visualizar en la pantalla todo el juego de caracteres:

```
10 FOR I=1 TO 255
20 PRINT I,CHR$(I)
30 A$=INKEY$:IF A$="" THEN 30
40 NEXT I
```

El programa visualiza un símbolo junto con un número. Pulse cualquier tecla para sacar otro número y su carácter en la pantalla. Mire cada número y observe su carácter correspondiente.

Si desea incorporar cualquier carácter al programa, con "PRINT CHR$ (170)" (o cualquier otro número) aparecerá el carácter en la pantalla. Los caracteres gráficos utilizan lo que MSX-BASIC llama "Screen 0".

Para hacer dibujos más complicados, necesitamos una resolución mayor en la pantalla para poder controlar elementos que sean más

pequeños que los caracteres completos. Esto nos permite dibujar con mucho más detalle que con los caracteres ASCII.

MSX-BASIC da acceso a la Pantalla 0 (Screen 0) además de otras tres pantallas los números 1, 2 y 3. La Pantalla 1 se utiliza como alternativa para la pantalla de texto y las Pantallas 2 y 3 para los gráficos en alta y baja resolución.

Para poder hacer uso de los mandatos de gráficos, debe Vd. antes indicar a su ordenador que desea utilizar una de las dos pantallas de gráficos. Si trata de utilizar un comando de gráficos con la pantalla de caracteres estándares, el ordenador producirá un mensaje de error. Si desea ver las diferencias que existen entre las dos pantallas, teclee el siguiente programa:

```
10 SCREEN 2
20 CIRCLE (125,100),50
30 GOTO 30
```

Lo que Vd. debe ver, en la pantalla es una aproximación bastante razonable de un círculo blanco en un fondo azul oscuro rodeado de un borde azul claro. Ahora ejecute el programa de nuevo, pero esta vez sustituya SCREEN 2 con SCREEN 3. Con ello, el círculo no solamente se dibuja con una línea mucho más gruesa que en el caso anterior sino que además se parece mucho menos a un círculo. La causa de esta diferencia está en la resolución de la pantalla.

La Pantalla 2 (SCREEN 2) se parece bastante a un papel milimetrado de 256 cuadrículas en sentido horizontal y 192 en sentido vertical (lo cual da una resolución de 49.152 cuadrículas. Dado que la pantalla de un televisor está razonablemente compactada, las cuadrículas, o **pixels**, son muy pequeñas. Es posible **direccionar**, o cambiar el color de cualquier pixel de la pantalla y ésta es la característica que le permite dibujar círculos, líneas y todas las figuras de las que hablaremos más adelante. Se puede cambiar el color de un pixel dado utilizando el mandato PSET. Todo lo que Vd. tiene que especificar es el pixel de la pantalla que Vd. quiere cambiar, mediante sus coordenadas x e y (el punto situado a 100 pixels a partir de la izquierda de la pantalla y a 50 pixels hacia abajo tiene como coordenadas 100, 50. Como demostración de cómo trabaja esto, teclee el siguiente programa:

```
10 SCREEN 2
20 PSET(120,20)
30 PSET(121,30)
```

```
40 PSET(122,40)
50 PSET(123,50)
60 PSET(124,60)
70 PSET(125,70)
80 PSET(126,80)
90 PSET(127,90)
100 PSET(128,100)
110 GOTO 110
```

Como puede ver, el programa ha cambiado el color de una serie de pixels, empezando en la posición 120,20 y terminando en la 128,100. Ahora sustituya SCREEN 2 por SCREEN 3 ejecute otra vez el programa y vea lo que ocurre.

La diferencia más obvia, como vio Vd. cuando el ordenador dibujaba el círculo, es que los pixels se dibujan de una forma bastante distinta si se utiliza la sentencia SCREEN 3 en lugar de SCREEN 2. En lugar de cambiar el color de un sólo pixel, la sentencia PSET hace que se cambie el color de un bloque de 4 X 4 pixels. Como Vd. puede ver, una coordenada horizontal de 120, 121, 122 ó 123 no produce ninguna diferencia sobre la posición horizontal en la que se dibuja el bloque. Cuando se llega al 124, se dibuja el bloque en la siguiente posición horizontal y ya no vuelve a modificarse hasta el 128.

Al igual que con SCREEN 2, SCREEN 3 tiene 256 X 192 pixels direccionables pero sólo puede dibujar en 64 X 48 bloques de 4 pixels (lo cual da una resolución de 3.072 bloques). Debe tener esto en mente cuando utilice SCREEN 3; de no ser así es muy posible que llegue Vd. a resultados verdaderamente confusos.

CIRCLE (<coordenada x>, <coordenada y>), <radio> [, <color>] [, <ángulo de comienzo>] [, <ángulo final>] [, <aspecto>]

El mandato CIRCLE tiene la sintaxis más complicada de las que hemos visto hasta ahora, por lo que vamos a dedicar algo de tiempo a explicar qué es exactamente lo que hacen sus elementos. Como Vd. ha visto en el programa anterior se consigue dibujar un círculo usando sólo los tres primeros elementos de la sintaxis. Los dos primeros definen la posición del centro del círculo (distancia desde el margen izquierdo y distancia desde el margen superior). El tercer elemento define el radio del círculo. Con sólo cambiar los valores del programa anterior verá cómo se mueve el círculo de la pantalla y cómo se cambia su tamaño.

Mediante el cuarto elemento, se puede cambiar el color con el círculo, así por ejemplo:

```
CIRCLE (125, 100), 50, 10
```

producirá un círculo amarillo en lugar de uno blanco (vea en el mandato COLOR la lista completa de los colores disponibles). Si Vd. sólo desea dibujar una parte del círculo, entonces debe utilizar los dos siguientes elementos de la sintaxis. El rango de ambos elementos puede variar entre –2xPI y 2xPI. PI es la relación que existe entre la circunferencia de un círculo con su radio (¡cómo aprendimos en la escuela!) y es aproximadamente igual a 3,142, así que el rango de valores aceptado por el ordenador es de –6,284 a +6,284. En lugar de tratar de explicar con detalle los efectos que tiene la variación de estos dos elementos de la sintaxis, le sugerimos que juegue con ellos y compruebe Vd. mismo lo que sucede. Cuando crea que ya ha practicado suficiente, intente ejecutar este programa que es una variación del primero que utilizamos para presentar el mandato CIRCLE:

```
10 SCREEN 2
20 LET X=0:LET Y=-.1
30 CIRCLE(125,100),90,7,Y,X
40 LET X=X+.1:LET Y=Y-.1
50 IF X<6.2 THEN GOTO 30
60 GOTO 60
```

Ahora ejecute otra vez el programa pero cambiando la línea 20 de X = 0 a X = 0.1.

La razón por la que hemos incluido el programa es para demostrar lo rápidamente que se puede apreciar la flexibilidad y la utilidad de un comando que se reconfigura totalmente al especificar más elementos de su sintaxis o al añadir un par de líneas extra de mandatos. No esperamos que comprenda el programa en este momento, es más importante que vea lo flexible que es el mandato CIRCLE. Por último, el parámetro aspecto. Es la relación entre la altura vertical del círculo y su anchura horizontal, y le permite dibujar una gran variedad de elipses. De nuevo, no es necesario seguir con explicaciones, es mucho mejor que haga pruebas cambiando este valor. Dicho esto, puede ser interesante añadir una X más al programa anterior convirtiendo la línea 30 en:

```
30 CIRCLE (125, 100), 90, 7, Y, X, X
```

Sencillo ¿verdad?

COLOR [<color de primer plano>], [<color de fondo>], [<color de borde>]

Vea la sección anterior, "Mandatos Interpretados".

DRAW, <Expresión tipo cadena>

DRAW, como PLAY (que veremos más adelante), utiliza un macro-lenguaje potente, el Macro-Lenguaje de Gráficos. DRAW le permite dibujar líneas coloreando determinados puntos de la pantalla (pixels). El Macro-Lenguaje de Gráficos utiliza una fórmula abreviada para dirigirse a los pixels que se quieren colorear.

U para arriba (up), D para abajo (down), L para izquierda (left) y R para derecha (right). Vd. puede dibujar, por ejemplo, un cuadrado utilizando un mandato que colorea 100 pixels hacia la derecha, luego 100 hacia abajo, luego 100 hacia la izquierda y sube, por último 100 pixels. El programa que hace esto es el siguiente:

```
10 SCREEN 2
20 DRAW "R100D100L100U100"
30 GOTO 30
```

Desde luego, esto es lo más sencillo que se puede hacer con el Macro Lenguaje de Gráficos. En el Capítulo 6 podrá ver Vd. toda la flexibilidad del lenguaje.

LINE (<x1, y1>) – (<x2, y2>), [<color>] [,<B/BF>]

El mandato LINE traza una línea entre dos puntos especificados. Como ejemplo, teclee:

```
10 SCREEN 2
20 LINE(20,10)-(240,180)
30 GOTO 30
```

Como puede ver, ha aparecido una línea recta entre los puntos 20, 10 y 240, 180. Se puede obtener esa línea en color añadiendo un número comprendido entre 0 y 15, así cambiando la línea 10 a:

```
20 LINE (20,10) – (240,180), 10
```

se obtiene una línea amarilla.

Con LINE también se puede dibujar un rectángulo que contiene a dicha línea como diagonal (añada el parámetro B a la línea 20). Se puede colorear el interior del rectángulo, (cambie la B de la línea 20 por BF) por lo que:

```
20 LINE (20, 10) – (240, 180), 10 BF
```

cambia la línea por un rectángulo amarillo.

LOCATE <x, y>

El mandato LOCATE se puede utilizar en ambas pantallas, de gráficos y de caracteres, para localizar la posición a partir de la cual se va a empezar a dibujar o a visualizar un texto. Como ejemplo, teclee el siguiente programa:

```
10 SCREEN 0
20 LOCATE 20,10
30 PRINT "Hola"
```

Cuando se ejecute el programa aparecerá la palabra "Hola" a partir de la posición 20 de la línea 10. Ocurre lo mismo con SCREEN 1. Para ver lo que sucede con la pantalla de gráficos hay que añadir un par de líneas al programa que quedará así:

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 LOCATE 20,10
40 PRINT#1, "HOLA"
50 GOTO 50
```

(Las líneas añadidas son necesarias para que Vd. pueda visualizar caracteres de texto en la pantalla de gráficos). Ahora ejecute otra vez el programa. De nuevo, el ordenador visualiza la palabra "Hola" a partir de la posición 20, 10 y como esta pantalla puede definir 256 × 192 posiciones en lugar de 40 × 23 de la pantalla 0 resulta que la palabra "Hola" queda desplazada hacia arriba y hacia la izquierda. Observe también que ha variado el tamaño de los caracteres ya que la pantalla de gráficos en alta resolución define sus caracteres de forma distinta a como lo hace la Pantalla 0.

Como demostración final cambie la SCREEN 2 de la línea 20 a SCREEN 3. Sale algo distinto ¿verdad?

PAINT (<x, y>) [,<color de pintura>] [,<color de borde>]

El comando PAINT se utiliza para rellenar una figura cualquiera de la pantalla de gráficos con el color de pintura especificado. En el siguiente Capítulo, veremos con más detenimiento cómo funciona este mandato, pero esencialmente, lo que hace PAINT es rellenar cualquier figura que Vd. haya dibujado en la pantalla, con el color que Vd. elija, empezando por un punto especificado que debe estar dentro de la figura. Para la demostración de este mandato, nos hemos visto obligados a presentar un programa algo más largo de lo habitual pero los resultados están llenos de colorido y compensan el esfuerzo realizado para teclearlo.

```
10 SCREEN 2
20 LET C=2
30 FOR 1=100 TO 1 STEP -10
40 CIRCLE (125,100),I,C
50 CIRCLE (125,100),C
60 LET C=C+1
70 NEXT I
80 GOTO 80
```

Como puede Vd. ver, el programa está dibujando círculos decrecientes, coloreando cada uno de ellos con un color, empezando en el centro. Es posible que Vd. pueda deducir del programa porqué se produce este efecto, pero le insistimos de nuevo: si Vd. no lo consigue, no se preocupe, seguro que todo estará claro al final de la siguiente sección.

POINT (<x, y>)

Con el mandato POINT Vd. puede establecer el color de un determinado pixel de la pantalla. Realmente POINT sólo es necesario para la programación avanzada de gráficos por lo que no vamos a complicarle las cosas intentando describirlo con mayor detalle, en este momento.

PSET (<x, y>) [,<color>]

Mientras que con el mandato PAINT Vd. puede dar color a una figura o bloque completo, con el mandato PSET puede especificar el color de un sólo pixel. Si, por ejemplo, ejecuta el siguiente programa:

```
10 SCREEN 2
20 PSET (100,100)
30 GOTO 30
```

verá que el pixel de la posición 100,100 se ha cambiado al color de primer plano. Si modifica la línea 20 y escribe:

```
20 PSET (100,100), 10
```

el punto se hace amarillo. Al igual que con los otros mandatos de gráficos al cambiar de SCREEN 2 a SCREEN 3 se producirá un cuadrado amarillo y así sucesivamente.

PRESET (<x, y>), <nombre del array> [,<opción>]

PRESET es, justamente, el opuesto de PSET ya que si no se especifica ningún color, PRESET dibujará el/los pixels en el color de fondo, en lugar del color de primer plano como sucede con PSET. Si se especifica algún color, entonces PRESET funciona exactamente igual que PSET.

PUT SPRITE <plano del duendecito> [,<x, y>] [,<color>] [,<n>]

Un duendecito (en inglés sprite) es un carácter de gráficos que se puede mover por toda la pantalla. Los duendecitos resultan familiares para cualquiera que haya jugado alguna vez a los Invasores del Espacio, o cualquier otro juego de vídeo similar, ya que los invasores, y la mayoría de las cosas que se mueven y atraviesan la pantalla se dibujan mediante los duendes.

Vd. ya está habituado al hecho de que la pantalla de gráficos está formada por 256 X 192 pixels. Los duendecitos se definen en pantalla y pueden tener cualquier contorno y tamaño, pudiendo variar éste de 1 a 32 X 32 pixels. Puede tener definidos hasta 256 en el programa, de los cuales 32 pueden aparecer en la pantalla, simultáneamente, sobre 32 planos separados ¡más que suficiente incluso para el diseñador más entusiasta de juegos!

La figura del duendecito se define utilizando la variable SPRITE$ y los atributos se definen con el mandato PUT SPRITE. La sintaxis de este mandato da el plano de la pantalla, que puede variar de 0 a 31, en el que se va a dibujar el duende; la posición en la que va a aparecer (la coordenada x se puede variar de –32 a +255; la coordenada y de –32

a +191, para utilización normal) y un número de paterna para el duendecito (debe ser menor de 256 si el duendecito se dibuja con 8 × 8 pixels y menor de 64 si se dibuja con 32 × 32 pixels). Por ejemplo:

```
PUT SPRITE 0,(100,100),7,17
```

produce la aparición de un duendecito, con número de paterna 17, de color azul grisáceo en el plano cero en la posición 100,100. (Se supone, claro está, que se ha definido previamente el duendecito 17).

NOTA: El mandato PUT SPRITE hace uso de lo siguiente: Si las coordenadas x e y se especifican como STEP (x, y) en lugar de simplemente como x e y, los valores de x e y para el movimiento del duendecillo son relativos al último punto en el que ha aparecido, así STEP (100,50) hace aparecer el duende 100 pixels hacia la derecha y 50 pixels hacia abajo del punto donde se encontraba, en lugar de situarlo en la posición 100,50. Si se especifica 208 para la coordenada y, desaparecen todos los planos situados detrás del que contiene el duendecito hasta que se asigne a y otro valor distinto de 208. Si se especifica 209 para y, desaparece el duendecito de la pantalla.

SCREEN [<modalidad>] [,<tamaño del duendecito>] [,<sonido de las teclas>] [,<velocidad del casete>] [,<opción de impresora>]

Suponemos que ya debe estar familiarizado con el mandato SCREEN. Sin embargo incluye una serie de características que no hemos tenido oportunidad de probar. Estamos seguros de que Vd. sabe perfectamente que la opción "modalidad" se utiliza para definir el tipo de pantalla que se desea utilizar, con las alternativas siguientes:

0 40 × 24 modalidad de texto
1 32 × 24 modalidad de texto
2 256 × 192 modalidad de gráficos en alta resolución
3 modalidad de gráficos en color en baja resolución

El tamaño del duendecito se puede seleccionar como:

0 8 × 8 sin ampliar
1 8 × 8 ampliado
2 16 × 16 sin ampliar
3 16 × 16 ampliado

Si a Vd. no le gusta el sonido de las teclas de su ordenador MSX, entonces seleccione la opción 0 para anularlo. Con la opción 1 las teclas vuelven a producir el click.

Los dos últimos elementos de la sintaxis le ayudarán en los problemas que pueda tener con una grabadora de casetes o con una impresora, si es que tiene una conectada al ordenador. Con el primero se puede establecer la velocidad, en baudios, con que se enviarán los programas a la grabadora de casete y puede ser 1 para 1.200 baudios ó 2 para 2.400 baudios. El último parámetro se utiliza si su impresor no se ajusta al estándar MSX debiendo ser distinto de cero, si éste es el caso.

SPRITE$ (<n>) = <expresión de tipo cadena>

En realidad SPRITE$ no es un mandato, es una variable. (Ver Capítulo 3 sobre variables y constantes).

Vamos a ver cómo se definen los duendes y cómo se escriben los programas que los utilizan en forma exhaustiva en el siguiente capítulo. Todo lo que necesita saber por el momento es que, para definir uno, todo lo que hay que hacer es asignar un nombre al duendecito y decir la forma que debe tener. Para esto existen varios métodos y los describiremos más adelante detalladamente. Por el momento, si está Vd. interesado en las distintas cosas que se pueden hacer con los duendecitos dedique un poco de tiempo al tecleo del siguiente programa:

```
10 SCREEN 2,0,0
20 FOR I=1 TO 8
30 READ B$
40 S$=S$+CHR$(VAL("&B$"+B$))
50 NEXT I
60 SPRITE$(0)=S$
70 X%=INT(RND(1)*256):Y%=INT(RND(1)*192)
80 PUT SPRITE 0,(X%,Y%),15,0:BEEP
90 FOR J=1 TO 300:NEXT J
100 GOTO 70
110 DATA 00011000
120 DATA 00111100
130 DATA 01100110
140 DATA 11011011
150 DATA 11011011
160 DATA 01100110
```

```
170 DATA 00111100
180 DATA 00011000
```

Observe cómo se define la figura del duendecito con los unos de las sentencias DATA.

VPEEK (<dirección de la RAM de vídeo>)

El mandato VPEEK le permite ver qué es lo que hay exactamente en un área determinada de la memoria de vídeo del ordenador. La utilización de los mandatos PEEK y POKE (que viene a continuación) están reservados para los programadores expertos y no están al alcance de los principiantes. En este momento, no es aconsejable que profundicemos más en este tema y no lo haremos.

VPOKE (<Dirección de la RAM de vídeo>), <byte>

Con VPOKE puede Vd. insertar un byte de información directamente en la memoria del ordenador; la dirección debe estar comprendida entre 0 y 16383. A menos que Vd. sepa perfectamente qué es lo que está Vd. haciendo le recomendamos encarecidamente que NO haga pruebas con este mandato (lo contrario de lo que le sugerimos con cualquier otro mandato del lenguaje). Si Vd. hace pruebas con el mandato POKE y el ordenador falla, ya no podrá hacerle trabajar. El problema se soluciona apagando y encendiendo otra vez el ordenador; de nuevo le advertimos en contra de su utilización.

En esta sección hemos intentado demostrar en detalle los más útiles, y por consiguiente los comandos más ampliamente utilizados que están disponibles en el MSX-BASIC y esperamos que Vd. haya encontrado interesante la experiencia de probarlos. Aunque los mandatos, por sí solos, son impresionantes, no desarrollan toda su potencia mientras no se combinan juntos en programas. Esperamos que esto le sea obvio por los programas que hemos incluido al objeto de clarificar las explicaciones. Aunque nuestra intención, como dijimos al principio del capítulo, era utilizar el menor número de programas posible, donde hemos tenido que utilizarlos esperamos que demuestren claramente que la inclusión de un par de líneas de un programa puede cambiar por completo la imagen que aparece en la pantalla.

Bueno, ya es hora de dar el paso que de la sencilla utilización de los mandatos, tal y como vienen en el sistema, le lleve a imponer al ordenador sus deseos mediante el arte de la programación del ordenador.

Empezar a programar

Después de ver tantos mandatos y sentencias disponibles en el MSX-BASIC ya es hora de que los pongamos en programas para hacer cosas de provecho. Rompiendo con una especie de tradición que parece ser una parte fundamental en cualquier libro de programación en BASIC, no le vamos a pedir que escriba programas para conocer cuál será el interés en su cuenta bancaria (ya tuvimos bastante con el ejemplo de la introducción), hacer que adivine sin ninguna razón, números aleatorios y cosas por el estilo. Por el contrario, hemos decidido utilizar al máximo las excelentes características gráficas de los ordenadores MSX, para ver cómo se pueden escribir programas para ver imágenes progresivamente más complejas e introducirle así en la programación estructurada de una forma más interesante de la habitual. Veamos cómo se pueden poner en práctica las ideas que ha aprendido en esta sección.

Cargar y salvar programas

Pero antes debemos aprender unos cuantos mandatos más, que son necesarios para realizar una serie de tareas muy frecuentes como son cargar y salvar los programas a/desde el ordenador. A medida que los programas se van haciendo cada vez más largos, es terrible tener que teclearlos de nuevo para cada nueva ejecución del programa o tener que listarlos después de cada ejecución satisfactoria. Si Vd. ha conectado a su ordenador una grabadora de casetes o una unidad de disco, entonces debe leer los siguientes mandatos, ya que tratan sobre este problema:

SAVE " <descriptor de dispositivo>: [<nombre del fichero>] "

El mandato SAVE salva el programa BASIC que está cargado en memoria en ese momento, en el dispositivo especificado por el descriptor de dispositivo, que puede ser CAS o DISC. Por ejemplo, SAVE "CAS:ANA" salvaría un fichero en el casete, y lo llamaría ANA.

LOAD "<descriptor de dispositivo : [<nombre del fichero>]" > [, R]

Este mandato se utiliza para cargar en memoria un programa BASIC desde casete. LOAD borra el programa que estuviese cargado en memoria. Si se especifica la opción R, el programa que se acaba de cargar empieza a ejecutarse automáticamente (RUN). Si se omite el nombre del fichero, se cargará el siguiente programa que se encuentre en la casete.

MERGE "<descriptor de dispositivo : [<nombre del fichero>]"

El mandato MERGE permite entremezclar programas. Es importante que los programas no contengan números de líneas conflictivas (por ejemplo, dos programas con número de línea 20 en ambas), ya que sólo se mantiene la línea que se está intercalando en memoria. El mandato funciona intercalando un programa BASIC, que está en casete (el casete es el medio, asumido por defecto, utilizado para el almacenamiento de los programas) en el programa que ya está cargado en memoria, por ejemplo MERGE "CAS:ANA" mezclará un programa llamado ANA, almacenado en casete, con el que está actualmente cargado en memoria. Si se omite el nombre del programa, se mezcla el siguiente programa que se encuentre.

BSAVE "<descriptor de dispositivo> : [<nombre del fichero>] " [,dirección de comienzo] [, dirección de ejecución]

A medida que vaya Vd. cogiendo experiencia en la programación seguramente dejará de programar en BASIC y seguirá con la programación en un nivel de programación más complejo.

BSAVE se utiliza para salvar un programa en código máquina. Se diferencia del mandato SAVE (utilizado para salvar programas en BASIC) en que puede grabar cualquier cosa que esté en memoria (incluso datos), en casete. La B de BSAVE significa "binario" ya que son datos binarios lo que se lee directamente de la RAM para su grabación. Su utilización, por lo tanto, está fuera del ambiente de este libro.

BLOAD "<descriptor de dispositivo> : [<nombre del fichero>]" [,R] [, <desplazamiento>]

El mandato BLOAD carga un programa en lenguaje máquina directamente en memoria y de nuevo decimos que está fuera del ambiente de este libro. Para completar la información añadiremos que la opción [, R] hace que el programa se ejecute inmediatamente después de su carga.

Escribir programas

Habiendo visto cómo almacenar y recuperar programas, ya es hora de aprender a escribir programas. Empezaremos por ver como se pueden

estructurar los programas para hacerlos más eficientes y además, para ahorrarle una gran cantidad de tecleo.

FOR...NEXT

A Vd. ya le son familiares los programas cortos como el siguiente:

```
10 SCREEN 2
20 CIRCLE (125,100),100
30 GOTO 30
```

Este programa dibuja un círculo con un radio de 100 pixels centrado en el punto 125,100. Pero ¿qué hay que hacer si se quieren dibujar una serie de círculos decrecientes? Bien, una forma (no necesita escribir éste) sería tecleando el siguiente programa:

```
10 SCREEN 2
20 CIRCLE (125,100),100
30 CIRCLE (125,100),90
40 CIRCLE (125,100),80
50 CIRCLE (125,100),70
60 CIRCLE (125,100),60
70 CIRCLE (125,100),50
80 CIRCLE (125,100),40
90 CIRCLE (125,100),30
100 CIRCLE (125,100),20
110 CIRCLE (125,100),10
120 GOTO 120
```

Como puede ver, lo que este programa hace es pedirle al ordenador que dibuje primero un círculo de 100 pixels de radio, después otro de radio 90 pixels, después de 80, y así hasta llegar a la línea 110 en la que se dibuja un círculo de 10 pixels. La última línea del programa incluye la sentencia GOTO que veremos en los siguientes programas y que estudiaremos con más detalle. En este caso particular, el GOTO sencillamente dice al ordenador que siga yendo a la línea 120 en lugar de finalizar la ejecución del programa. Si no se hubiese especificado la sentencia GOTO, los círculos desaparecerían de la pantalla y el ordenador retornaría a la pantalla de caracteres tan pronto como se hubiese dibujado el último círculo.

Ahora bien, teclear todo lo anterior es algo extremadamente aburrido (fue peor para mí tener que teclearlo que para Vd. leerlo), además de

innecesario ya que todo lo que tiene que hacer para alcanzar el mismo resultado es incluir un bucle FOR...NEXT en el programa. Hablaremos bastante de bucles en las siguientes páginas, y esencialmente lo que los bucles hacen es conseguir que el ordenador realice la misma acción un número de veces, ya sea de la misma forma o de forma ligeramente diferente. Para demostrar cómo funciona un bucle FOR...NEXT, teclee lo siguiente:

```
10 SCREEN 2
20 FOR N=10 TO 100
30 CIRCLE (125,100),N
40 NEXT N
50 GOTO 50
```

Lo que el ordenador ha hecho ha sido dibujar una serie de círculos, empezando por un radio de 10 y terminando con un radio de 100, centrados en el punto 125, 100. ¿Cómo lo ha conseguido? Bien, la primera vez que el ordenador trata la línea 20 la sentencia FOR le dice que asigne el valor 10 a una variable de nombre N. Por tanto, cuando ejecuta la línea 30, sabe que tiene que dibujar un círculo de 10 pixels de radio. La línea 40 le dice al ordenador que tome el siguiente valor de N. A continuación vuelve a la línea 20 y como N se ha especificado para que valga de 10 a 100, se le añade 1 al valor existente y en la línea 30 dibuja un círculo con un radio de 11 pixels. Esto continúa así hasta que N alcanza el valor 100, en ese punto el ordenador lee la línea 50 que le dice que se quede donde está.

Pero volviendo a nuestro problema original, lo que nosotros queríamos era una serie de círculos, cuyos radios varían de 10 en 10. Para conseguir esto se cambia la línea 20 de forma:

```
20 FOR N = 10 TO 100 STEP 10
```

Al introducir STEP en el mandato, le hemos dicho al ordenador que incremente N no en 1 sino en un valor que hemos especificado, en este caso 10. Así pues, el ordenador dibujará ahora círculos con radios de 10, 20, 30...100 pixels con el resultado final que queríamos que produjese el programa original.

La forma en que se dibujan los círculos, sin embargo, no es del todo correcta. En nuestro programa original (y largo) el ordenador dibujaba círculos cuyos radios empezaban en 100 y terminaban en 10. El programa actual hace exactamente lo contrario. El problema se resuelve si cambiamos de nuevo la línea 20 y escribimos:

```
20 FOR N = 100 TO 10 STEP –10
```

Los círculos se dibujan ahora utilizando tres líneas de programa en lugar de las 10 que habíamos visto que se necesitaban originalmente. Si volvemos al programa que se utilizó para demostrar el mandato PAINT, podrá Vd. ver exactamente cómo se van rellenando los círculos con sus colores respectivos. El programa que usamos entonces era el siguiente:

```
10 SCREEN 2
20 LET C=2
30 FOR N=100 TO 0 STEP -10
40 CIRCLE (125,100),N,C
50 CIRCLE (125,100),C
60 LET C=C+1
70 NEXT N
80 GOTO 80
```

La estructura de este programa es la misma que la del que acabamos de ver, con la única excepción de que se ha incluido el mandato PAINT de forma que también hace uso del bucle FOR...NEXT. Como puede ver, el color de pintura es variable (C). Inicialmente el color establecido es el 2 (en la línea 20), y cada vez que se ejecuta el bucle FOR...NEXT, el número de color se incrementa en 1 (línea 60). Esto se consigue utilizando la sentencia LET. También habrá observado que el color con el que se dibuja el círculo es el mismo que el utilizado para rellenarlo. Permítanos volver a nuestro problema de dibujo de círculos para añadirle un par de líneas:

```
10 SCREEN 2
15 FOR M=60 TO 180 STEP 40
20 FOR N=100 TO 10 STEP -10
30 CIRCLE (M,100),N
40 NEXT N
45 NEXT M
50 GOTO 50
```

Todo lo que hemos hecho aquí ha sido añadir otra variable en el mandato CIRCLE y, utilizando otro bucle FOR...NEXT, pedir al ordenador que mueva la coordenada x del centro de los círculos, 40 pixels hacia la derecha cada vez que dibuja uno. Observe que los bucles no deben cruzarse, el bucle FOR M...NEXT M encierra completamente al bucle FOR N...NEXT N. Siempre que escriba un programa debe estructurar

los bucles de esta forma, de lo contrario el ordenador le responderá con un mensaje de error.

Añadamos este segundo bucle FOR...NEXT a nuestro programa para dibujar círculos y después colorearlos. Todavía no le hemos pedido que haga nada Vd. sólo, así que intente incluir este segundo bucle sin mirar al programa que viene a continuación.

```
10 SCREEN 2
20 LET C=2
30 FOR M=60 TO 180 STEP 40
40 FOR N=100 TO 10 STEP -10
50 CIRCLE (M,100),N,C
60 PAINT (M,100),C
70 LET C=C+1
80 NEXT N
90 NEXT M
100 GOTO 100
```

Su programa debería parecerse bastante al nuestro (nosotros lo hemos RENUMerado sobre la marcha para evitar que las cosas se amontonen). Si es así, nuestras felicitaciones, debe estar contento de Vd. mismo. Si no es así, no se preocupe demasiado ¡sólo es cuestión de práctica! Ahora ejecutemos el programa y veamos qué sucede ¡el resultado final debe estar lleno de color!

¿Qué es lo que está mal? Como puede ver, el ordenador se ha quedado a medio camino cuando estaba con el segundo juego de círculos, se ha parado y ha producido un mensaje de error:

```
Illegal function call in 50
```

Analice la línea 50 y trate de descubrir lo que ha sucedido.

La respuesta es que el ordenador se ha salido del rango de colores con los que poder rellenar los círculos. Los números de colores disponibles varían del 0 al 15 solamente. Al arrancar el programa le dijimos al ordenador que empezase con el color 2. Catorce círculos después, lógicamente, alcanzó el color 15. Cuando se llega a la línea 70 este número de color se incrementa a 16 de forma que cuando el ordenador vuelve a la línea 50 otra vez, se le pide que dibuje un círculo en un color que no entiende, de ahí el mensaje de error.

Así que ¿cómo podemos arreglar el programa para completar su paterna de círculos?

IF...THEN

La razón por la cual nuestro programa ha "fallado" y nos ha dado un error era que permitíamos que el número del color se saliese de su rango establecido, y le pedíamos al ordenador que hiciese algo que no entendía, dibujar un círculo usando el color 16, que no existe. La solución consiste en evitar que el número del color pueda sobrepasar su límite superior mediante la verificación del número y diciéndole al ordenador qué es lo que debe hacer cuando se alcance ese límite. Y esto es precisamente lo que se consigue al añadir la siguiente línea al programa:

```
75 If C = 15 THEN C = 2
```

Con esto se verifica si C vale 16 y si es así, se restaura su valor a 2. Si no lo es, el programa continúa con la línea 80.

Si ejecuta el programa de nuevo, incluida esta línea, verá que esta vez todo funciona según lo previsto. El programa se ejecuta igual que la vez anterior, pero esta vez con la diferencia de que cuando en la línea 70, C alcanza el valor 16, la línea 75 lo detecta y hace que C vuelva a valer 2, permitiendo que el proceso vuelva a empezar. Esto mismo ocurre cada vez que C alcance el valor 16 hasta que se hayan dibujado y rellenado todos los círculos.

En el ejemplo anterior hemos utilizado el IF...THEN para salir de un problema en el que nos habíamos metido. Son muchas las ocasiones en las que se puede utilizar el IF; dejamos a su imaginación la decisión de cómo y cuándo emplearlo. Vd. tiene dos opciones en lo que respecta a lo que puede pedir que haga el ordenador SI (IF) algo sucede.

IF...THEN...ELSE

Volviendo a nuestro programa de dibujar círculos, todo lo que nos interesaba era comprobar si C era igual a 16 y si lo era, cambiar su valor a 2. Si la condición es "verdadera" ("true") entonces se ejecuta la línea; si es falsa, por ejemplo C no es igual a 16, entonces el ordenador pasa a la línea siguiente. Con programas mayores que éste sería más conveniente, si la condición es falsa, pasar a otra línea del programa distinta de la siguiente, siendo ésta una forma de utilización de IF...THEN...ELSE. Por ejemplo:

```
IF C = 16 THEN C = 2 ELSE 80
```

es una forma alternativa de escribir la línea 75. El resultado obtenido después de la ejecución de la línea es el mismo, pero en lugar de dejar

que el ordenador ejecute la línea 80, simplemente porque es la siguiente línea del programa, le está Vd. indicando exactamente qué es lo que tiene que hacer si la condición es falsa.

Nota: Los números de línea y las instrucciones de programación son igualmente válidas cuando se escriben detrás de THEN o ELSE, como queda demostrado en el cambio que hemos hecho en la línea 75.

IF...GOTO

Vd. ya ha visto la sentencia GOTO en todos los programas que le hemos pedido que teclee. Hasta el momento, sólo hemos utilizado el GOTO para asegurarnos de que el resultado de la ejecución del programa permanece en la pantalla haciendo en la última línea del programa un bucle que vuelve sobre la misma línea hasta que Vd. lo detecta mediante las teclas CONTROL y STOP. Lo que hace el GOTO, sencillamente, es hacer que el programa salte al número de línea indicado por él.

Para demostrar cómo funciona el GOTO, teclee el siguiente programa y ejecútelo. Antes de leer la explicación de lo que va a suceder, estudie un poco el programa y vea si lo puede deducir por Vd. mismo. Es el programa más complicado de los que hemos mostrado, así que no se preocupe si lo encuentra un poco difícil. Esencialmente, el programa es el mismo que el programa de dibujar círculos; sólo le hemos añadido unas pocas líneas para que el ordenador le pregunte si Vd. desea que el programa se ejecute o no. Cuando arranque el programa, conteste *sí* o *no* cuando aparezca la pregunta en pantalla pero pruebe a contestar *bueno* o cualquier otra palabra por el estilo y vea lo que sucede entonces.

```
10 CLS
20 PRINT "Quiere que dibuje algunos
círculos?"
30 PRINT "Por favor, conteste sí o no"
40 INPUT A$
50 IF A$="sí" GOTO 90
60 IF A$="no" GOTO 170
70 PRINT "Lo siento, no comprendo.
Pruebe otra vez"
80 GOTO 40
90 SCREEN 2
100 LET C=2
```

```
110 FOR N= 100 TO 10 STEP -10
120 CIRCLE (125,100),N,C
130 CIRCLE (125,100),C
140 LET C=C+1
150 NEXT N
160 GOTO 160
170 END
```

Veamos qué es lo que hace el programa línea a línea.

La línea 10 limpia la pantalla.

La línea 20 le dice al ordenador que visualice en la pantalla la pregunta "¿Quiere que dibuje algunos círculos?"

La línea 30 le está pidiendo que responda escribiendo "sí" o "no".

La línea 40 hace que el ordenador se mantenga a la espera hasta que Vd. haya escrito una palabra. Asigna una variable llamada A$ para almacenar su respuesta y visualiza el signo ? en la pantalla para hacerle saber que está esperando algún tipo de contestación.

La línea 50 le dice al ordenador que si Vd. responde "sí" entonces debe pasar a la línea 90, ignorando las líneas 60, 70 y 80. A continuación se ejecuta nuestro conocido programa de los círculos. Si su respuesta no fuese "sí" entonces el GOTO de la línea 50 no se ejecuta y el ordenador pasa a la línea 60.

La línea 60 le dice al ordenador que si Vd. responde "no" debe ir a la línea 170 donde la sentencia END produce el cese de la ejecución del programa y devuelve el mensaje de diálogo OK a la pantalla.

La línea 70 le dice al ordenador que visualice en la pantalla el mensaje "Lo siento. No entiendo. Pruebe de nuevo", si Vd. ha contestado algo distinto de "sí" o "no" en la línea 40.

La línea 80 le dice al ordenador que vaya otra vez a la línea 40 donde debe quedar a la espera de una respuesta que él pueda comprender.

Esperamos que con estas explicaciones vea Vd. todo lo flexible que pueden ser las sentencias GOTO y la cantidad de tecleo que pueden ahorrar permitiendo que Vd. utilice la misma parte del programa una y otra vez, de forma muy similar a los bucles FOR...NEXT.

IF...GOTO...ELSE

Las sentencias IF...GOTO...ELSE funcionan exactamente de la misma forma que las IF...THEN...ELSE, si bien el GOTO limita la acción después del IF a un salto a una línea diferente.

Por último, es totalmente correcta una línea de mandato combinando una serie de estas comprobaciones condicionales, como por ejemplo:

IF...THEN IF...AND...GOTO...ELSE

o incluso

IF...OR IF...AND. IF...THEN...ELSE

Los programas pueden ser tan complejos como Vd. desee hacerlos. No vamos a intentar mostrarle un programa con este tipo de sentencias, por el momento. Preferimos decirle sencillamente que a medida que vaya Vd. incrementando su habilidad en la programación, y que los programas que Vd. sea capaz de escribir vayan complicándose, rápidamente empezará Vd. mismo a pensar en estas sentencias.

Habrá Vd. observado que hemos introducido una serie de nuevos mandatos en el último programa llamados PRINT, INPUT y END que no habíamos visto antes. Como mencionamos al principio del capítulo, algunas instrucciones sólo tienen significado real si se utilizan en programas y éstas, junto con las sentencias FOR e IF, pertenecen a este grupo. Resulta adecuado, llegados a este punto en el que Vd. ya está familiarizado con los elementos básicos de la construcción de programas, que veamos con detalle estas sentencias.

Sentencias de programas

Las sentencias que aparecen en esta sección no tienen una sintaxis como tal. En su lugar, el nombre de la sentencia va seguido por una determinada información necesaria para que la sentencia realice su trabajo.

DATA <lista de constantes>

En todos los programas que hemos visto hasta ahora le hemos pedido al ordenador que realice procesos con cosas que él ya conoce, como cambiar el color o la anchura de un círculo. ¿Qué es lo que tenemos que hacer si queremos decirle el número de km recorridos por un coche con una cantidad dada de gasolina para que un programa calcule el consumo medio? Una solución consiste en la utilización de la sentencia DATA. En realidad, las sentencias DATA no hacen nada, únicamente dicen al ordenador que los números o las palabras que van a continuar

son datos a los que se puede acceder con el mandato READ (que veremos más tarde). Estos son ejemplos de sentencias DATA:

```
DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
DATA Enero, Febrero, Marzo, Abril, Mayo, Junio
```

Una sentencia DATA puede contener tantos números o palabras como quepan en una línea, cada uno de ellos separados por una coma, pudiéndose utilizar múltiples sentencias DATA en el mismo programa —se accede a los datos como si estuviesen en una larga lista, por eso no importa el lugar en el que se coloquen las líneas DATA en un programa siempre y cuando estén en el orden correcto. Se permite cualquier tipo de números, excepto las expresiones numéricas, como por ejemplo 2 + 3. Si se incluye una constante de tipo cadena en la que aparece una coma, los dos puntos o un espacio como *3 de Enero*, entonces hay que encerrarla entre comillas. (Una cadena es un grupo de caracteres como un nombre, un número de teléfono o una palabra).

Por último, los datos incluidos en la sentencia DATA deben ser del mismo tipo que el esperado por la sentencia READ que accede a ellos.

DIM <lista de variables subindicadas>

Con la sentencia DIM entramos en el tema de los arrays, ya que se utiliza para especificar el tamaño de un array en memoria. Un array es un tipo especial de variable que permite almacenar y acceder fácilmente a un grupo de variables relacionadas entre sí.

El array más sencillo consiste en una serie de variables, agrupadas bajo un sólo nombre del array, digamos "A". Al principio del programa debe Vd. definir el número de variables que va a almacenar en A. A esta operación se le denomina DIMensionar. Podemos establecer qu A va a albergar diez variables utilizando el mandato DIM A(10).

A cada variable se le da un número dentro del array, siendo accesible mediante el nombre del array y el número de la variable. Por ejemplo, A(3) es la tercera variable y A(7), la séptima.

Los arrays permiten la utilización de números como nombres de variables allí donde podría ser problemática la creación de nuevos nombres de variables. Si queremos almacenar y visualizar los cuadrados de los diez primeros números, podríamos tener diez sentencias del tipo LET A = 1 * 1, LET B = 2 * 2 y así sucesivamente. Esto no resulta nada práctico. Una forma más fácil para hacer esto consiste en la utilización de un bucle FOR...NEXT y un array en un programa como éste:

```
10 DIM A(10)
20 FOR I=1 TO 10
30 A(I)=I*I
40 PRINT I,A(I)
50 NEXT I
```

Si Vd. sigue todo el programa, empezando con I = 1, la primera sentencia establece A(1) = 1 * 1. La siguiente sentencia establece A(2) = 2 * 2 y así hasta 10. Ahora podemos manipular las variables de los arrays como queramos —podemos dividir cada una por 10 añadiendo la línea:

```
35 A(I) = A(I)/10
```

END

La sentencia END únicamente dice al ordenador que ha llegado al final del programa y le indica que debe finalizar la ejecución del programa, cerrar los ficheros que el programa haya creado y volver al nivel de mandato con la aparición del mensaje de diálogo OK en la pantalla.

ERROR <expresión entera>

El MSX-BASIC contiene un número de mensajes de error predefinidos, algunos de los cuales ya los hemos visto. El Apéndice B es una relación completa de estos mensajes. Cada mensaje lleva asociado un número comprendido entre 0 y 255. Como puede ver si mira el apéndice, los números 0 al 60 ya están asignados, lo cual le deja a Vd. los números 61 al 255 para sus propios mensajes. Es mejor que escriba sus mensajes empezando en 255 y seguir hacia abajo para asegurar la compatibilidad en el supuesto de una ampliación del número de mensajes de error contenidos en el MSX-BASIC. Un ejemplo de cómo puede incluir sus propios códigos de errores es el siguiente:

```
10 ON ERROR GOTO 1000
100 IF A$="No" THEN ERROR 250
1000 IF ERROR=250 THEN PRINT "Está Vd.
absolutamente seguro?"
```

FOR

Ver página 53.

GOSUB <número de línea>

Ya hemos visto cómo la sentencia GOTO hace que el ordenador salte a un número de línea especificado. La sentencia GOSUB es un tipo avanzado de GOTO ya que dice al ordenador que vaya a una subrutina. Una subrutina es una parte del programa que se escribe para un propósito determinado y que puede ser llamada múltiples veces desde el programa principal. Como ejemplo, si ampliamos el programa de dibujar círculos para incluir la pregunta "¿dibujar algunos cuadrados?" se puede utilizar esto como una respuesta si le contestó "no" a la pregunta de "¿dibujar algunos círculos?". Añada otra opción "¿ejecutar una tonada?" y verá como empieza a complicarse el programa. Para simplificar las cosas, si se dijo "sí" a alguna de las preguntas, entonces una sentencia GOSUB hace que el ordenador vaya a la subrutina que dibuja círculos o a la que dibuja cuadrados o a la que ejecuta una tonada.

Al final de una subrutina hay que incluir una sentencia RETURN para indicar al ordenador que debe volver a la línea siguiente a la sentencia GOSUB que es la que provoca la ejecución de la subrutina.

Sin escribir todas las líneas del programa, la estructura de nuestro programa super-ampliado, para dibujar círculos, utilizando subrutinas, sería la siguiente:

```
10 DEBO DIBUJAR ALGUNOS CIRCULOS?
20 IF "sí" GOTO 100
30 DEBO DIBUJAR ALGUNOS CUADRADOS?
40 IF "sí" GOTO 120
50 DEBO EJECUTAR UNA TONADA
60 IF "sí" GOTO 140
70 IF "no" GOTO 500
80 PRINT "Lo siento, no comprendo.
Pruebe de nuevo."
90 GOTO 10
100 GOSUB 200
110 GOTO 10
120 GOSUB 300
130 GOTO 10
140 GOSUB 400
```

```
150 GOTO 10
200 REM Subrutina de dibujar circulos
.
.
.
.
299 RETURN
300 REM Subrutina de dibujar cuadrados
.
.
.
.
399 RETURN
400 REM Subrutina de ejecutar una tonada
.
.
.
.
499 RETURN
500 END
```

Veremos la sentencia REM más adelante.

Nota: Es conveniente poner siempre las subrutinas al final del programa y entrar en ellas con el mandato GOTO, ya que con esto se evita que se pueda entrar a ejecutarlas inadvertidamente durante la ejecución normal del programa. Es necesario, sin embargo, parar el programa para que no vaya a ejecutar la rutina con el GOSUB, pero que dé el mensaje "Return without GOSUB" (return sin GOSUB) cuando detecte la sentencia RETURN y se encuentre con que no tiene lugar a dónde volver. En el último ejemplo hemos utilizado la sentencia GOTO 10 en la línea 90 para detener el programa.

GOTO <número de línea>

Ver página 58.

IF <expresión> THEN <expresión> ELSE <expresión>

Ver página 57.

INPUT ["<mensaje de diálogo>";] <lista de variables>

Como seguramente habrá visto con el programa en el que se le pedía que indicase si el ordenador debía dibujar algunos círculos, la sentencia INPUT se utiliza para indicarle al ordenador que es preciso realizar una entrada desde el teclado para proseguir con la ejecución del programa (en nuestro caso, esta entrada podría ser "sí" o "no"). Para que Vd. sepa que el ordenador está aguardando su respuesta, aparece en la pantalla un signo de interrogación que puede ir precedido por el mensaje de diálogo especificado, si se incluyó en la sentencia. En el último programa veíamos cómo utilizar un mensaje de diálogo con INPUT. En el programa, la respuesta se almacenaba en una variable llamada A$ y el ordenador sabía que dicha variable debía valer "sí" o "no", de lo contrario contestaría diciendo que no ha comprendido y le pediría que empezase de nuevo. Siempre que una sentencia INPUT le solicite una información su respuesta debe coincidir con la que el ordenador está esperando, de no ser así, contestará con un mensaje de error hasta que pueda aceptar la respuesta recibida.

Los mensajes de error asociados a las entradas incorrectas son como sigue:

Si se responde con un tipo de datos incorrecto, por ejemplo una letra en lugar de un número, el ordenador visualizará:

```
?Redo from start
```

Si se responde con más elementos de datos de los esperados, por ejemplo 1, 2, 3 cuando el ordenador sólo espera dos números, se obtendrá el mensaje:

```
?Extra ignored
```

Por último si se responde al INPUT con menos datos de los esperados, se obtendrá:

```
??
```

y el ordenador se queda a la espera de los datos que faltan.

Vd. puede salir de esta operativa del INPUT pulsando las teclas CONTROL y STOP simultáneamente. Tecleando CONT se reanuda el proceso en la sentencia INPUT.

LINE INPUT ["<mensaje de diálogo>";] <variable de tipo cadena>

LINE INPUT funciona de la misma forma que INPUT, con la excepción de que permite la asignación de una línea completa, hasta un máximo de 254 caracteres, a una variable de tipo cadena sin tener que utilizar las comillas. No aparece en pantalla el signo de interrogación a menos que Vd. lo especifique en el mensaje de diálogo. Todo lo que Vd. teclee a continuación del mensaje de diálogo quedará asignado a una variable de tipo cadena. La forma de salir y retornar al LINE INPUT es idéntica a la del INPUT.

[LET] <variable> = <expresión>

La sentencia LET asigna el valor de una expresión a una variable, por ejemplo LET C = 2, LET A$ = "sí", etc. Observe que la palabra LET es opcional, todo lo que necesita es el signo =, de esta forma las sentencias C = 2 ó A$ = "sí" son igualmente válidas.

LPRINT [<lista de expresiones>]
LPRINT USING <expresión de tipo cadena> ; <lista de expresiones>

Estas sentencias se utilizan para sacar los datos por una impresora de líneas. Para más detalles ver PRINT y PRINT USING.

MID $ (<expresión tipo cadena 1>, n [, m]) = <expresión tipo cadena 2>

MID$ le permite sustituir parte de una cadena con otra. Los caracteres de la cadena 1 son sustituidos por los de la cadena 2, empezando en la posición n de la cadena 1. Se puede utilizar m para especificar el número de caracteres de la cadena 2 que va a utilizarse en la sustitución. Es imposible sustituir más caracteres de los que existen en la cadena 1. Como ejemplo.

```
10 LET A$="Sonrisa"
20 LET B$="rojo no"
30 MID$(A$,4,4)=B$
40 PRINT A$
```

cambiará el valor de A$ de Sonrisa a Sonrojo.

NEXT <variable>

Ver página 53.

ON ERROR GOTO

En la sentencia ERROR ya pudo Vd. ver cómo funcionaba la sentencia ON ERROR GOTO. Esta sentencia le dice al ordenador qué es lo que tiene que hacer si se produce un error. Si Vd. no le dice al ordenador qué hacer en el programa, o si Vd. incluye la línea ON ERROR GOTO 0, el ordenador detendrá la ejecución del programa y escribirá el mensaje de error en la pantalla. Esto le da al programador un mayor control sobre los errores que se pueden producir en un programa, e incluso le permite la recuperación después de una serie de errores que normalmente habrían provocado la parada de un programa. Es preferible poner esta sentencia directamente al principio del programa.

ON <expresión> GOTO <lista de números de líneas>

ON ERROR GOTO es justamente un ejemplo de la sentencia ON... GOTO. Se utiliza de forma más generalizada para proporcionar un juego de GOTOs opcionales, eligiendo uno determinado dependiendo del valor de la expresión. Por ejemplo, si el valor de la expresión es 3, el programa saltará al tercer número de línea en la lista que hay después del GOTO.

Si el valor de la expresión es cero, o mayor que el número de elementos de la lista, pero menor o igual a 255, entonces se ignora el GOTO y se ejecuta la siguiente sentencia.

ON <expresión> GOSUB <lista de números de línea>

ON...GOSUB funciona de la misma forma que ON...GOTO con cada número de la lista de números de línea, pasando a la primera línea de la subrutina.

POKE <dirección de memoria>, <expresión entera>

POKE se utiliza de la misma forma que el mandato VPOKE que mostramos en la sección de mandatos de gráficos y debe emplearse con el

mismo tipo de precauciones. Sirve para poner un determinado valor en una posición específica de memoria.

<dirección de memoria> es la dirección de la posición de memoria que se va a modificar y debe estar comprendida entre –32.768 a +65.535. Si el valor es negativo se toma el resultado obtenido de restarlo de 65.536, es decir, –1 = 65.535.

<expresión entera> es el dato que se va a escribir y debe estar comprendido entre 0 y 255. En nuestra introducción, ya vio Vd. que la cantidad máxima de memoria accesible para su modificación es 64K (64K = 64 × 1.024 = 65.536 bytes).

PRINT ["<lista de expresiones>"]

La sentencia PRINT se utiliza para visualizar información en la pantalla del televisor como ya hemos visto varias veces con la versión final del programa de dibujar círculos. Si se utiliza PRINT sin la lista de expresiones, se visualiza una línea en blanco (con espacios). Si se incluye una expresión, ésta se visualizará, así por ejemplo PRINT "Hola" hará que la palabra Hola aparezca en pantalla.

Las posiciones en las que se visualizan los elementos dependerán de los signos de puntuación utilizados para separar los elementos de la lista, como indicamos a continuación:

- , las líneas de visualización se dividen en zonas de 14 espacios cada una. Una coma hace que el siguiente elemento aparezca al comienzo de la siguiente zona disponible.
- ; un punto y coma hace que el siguiente elemento se visualice inmediatamente después del anterior. El tecleo de uno o más espacios entre elementos tendría el mismo efecto.

Si se teclea una coma o un punto y coma al final de la <lista de expresiones>, la siguiente sentencia PRINT empezará la visualización en la misma línea, respetando los espacios. Si no se incluye la coma o el punto y coma, la siguiente sentencia PRINT, empezará a visualizar los datos en la siguiente línea.

Los números aparecen siempre seguidos de un espacio. Los números positivos también van precedidos por un espacio, mientras que los números negativos van precedidos por un signo –.

Para que Vd. no tenga que estar escribiendo continuamente la palabra PRINT puede, si quiere, sustituirla con un signo de interrogación ya que el MSX-BASIC le asigna el mismo significado que la sentencia PRINT.

Para las características anteriores de la sentencia PRINT, y, en la más pura tradición de los libros de ordenadores, ¿por qué no intenta lo siguiente?:

```
10 CLS
20 PRINT "Hola"
30 PRINT "Hola","Hola"
40 PRINT "Qué es";"lo que"
50 PRINT "pasa"
60 PRINT "aquí"
70 PRINT "entonces";
80 PRINT "?"
```

Un poco rebuscado ¿verdad?

Si Vd. intenta ejecutar este programa con alguna de las pantallas de gráficos, verá que no sucede nada. Para resolver este problema, hay que escribir un programa del siguiente tipo (como vio Vd. cuando estudiamos el mandato LOCATE).

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 PRINT #1,"Lo que Vd. quiera que
imprima el ordenador"
40 GOTO 40
```

PRINT USING <expresión de tipo cadena> ; <lista de expresiones>

Comparada con la facilidad de la utilización de la sentencia PRINT, PRINT USING es bastante complicada. Para ahorrarle complicaciones en este momento, lo hemos propuesto para un capítulo posterior.

READ <lista de variables>

Cuando hablamos de la sentencia DATA nos referimos a la sentencia READ y dijimos que hay que utilizar ambas conjuntamente. Lea otra vez lo que escribimos sobre la DATA y ejecute el siguiente programa:

```
10 PRINT "La suma de los números de";
20 PRINT " las sentencias";
30 PRINT " DATA es";
```

```
40 READ A,B,C,D,E,F
50 G=A+B+C+D+E+E+F
60 PRINT G
70 DATA 1,2,3
80 DATA 4,5,6
```

Excepcionalmente, hemos hecho este programa deliberadamente un poco más largo de lo que sería preciso para demostrar algunas de las cosas que hemos comentado sobre las sentencias PRINT y DATA, por ejemplo la utilización del punto y coma, el espacio que precede a los números cuando son visualizados y el hecho de que los datos se leen, bajo cualquier circunstancia, secuencialmente. ¡Si no se había percatado de esto, obviamente es que no le ha prestado ninguna atención! Observe también que aunque los datos pueden aparecer en cualquier parte del programa, los hemos puesto al final para que no se confunda con el resto del programa.

Una sentencia READ puede acceder a más de una sentencia DATA, varias sentencias READ pueden acceder a la misma DATA. Le demostraremos esto cuando estudiemos el mandato RESTORE así que ¡no borre el programa!

REM <observaciones>

REM se utiliza sencillamente para permitirle la introducción de observaciones o notas aclaratorias, dentro del programa, explicando qué es lo que hacen determinados segmentos del programa. Hemos utilizado REM en el programa de dibujar círculos super-ampliado para mostrar qué es lo que estaba haciendo cada subrutina. El ordenador ejecuta las sentencias REM pero, como Vd. puede ver por la forma en que se estructuró el programa, pueden utilizarse como puntos de entrada a las subrutinas vía GOSUB o desde una sentencia GOTO. Vd. puede añadir observaciones al final de una línea usando una ' o : REM, pero es preferible dedicar una línea completa para las observaciones porque de esta forma se sigue más fácilmente la estructura del programa.

RESTORE [<número de línea>]

Como apuntamos en la sentencia READ, para volver a leer los datos hay que restaurarlos. Si todavía mantiene en pantalla el programa que dábamos en la sección del READ, conviértalo en el siguiente y verá qué es lo que queremos decir:

```
10 PRINT "La suma de los números de";
20 PRINT " las sentencias";
30 PRINT " DATA es";
40 READ A,B,C,D,E,F
50 G=A+B+C+D+E+F
60 PRINT G
70 RESTORE
80 PRINT "El producto de los números
es";
90 READ A,B,C,D,E,F
100 H=A*B*C*D*E*F
110 PRINT H
120 DATA 1,2,3
130 DATA 4,5,6
```

Si Vd. intenta ejecutar el programa sin la sentencia RESTORE aparecerá el mensaje "Out of DATA in 90". Observe también que se puede asignar un número de línea al RESTORE, en este caso sólo se pueden volver a leer los datos situados después de dicho número de línea; con RESTORE 120 podríamos ejecutar el programa anterior mientras que con RESTORE 130 no.

Nota: Habrá observado que las líneas 70 y 90 del programa anterior no son necesarias. Habiendo asignado los valores a A,B,C,D,E y F no hay ninguna necesidad de leer los datos de nuevo (línea 90). La razón por la cual se ha escrito este programa así es porque queríamos mostrar, de la forma más sencilla posible, cómo funciona la sentencia RESTORE. La encontrará incluida en otros programas del libro (más adelante) más complejos, en los que sí es necesaria.

RESUME

La sentencia RESUME le dice al ordenador qué es lo que debe hacer después de que se haya realizado la recuperación de un error. Se pueden utilizar cuatro formatos para el RESUME, dependiendo de donde se quiera que prosiga la ejecución del programa:

RESUME o **RESUME 0** hace que la ejecución se reanude en la sentencia que provocó el error.

RESUME NEXT hace que la ejecución se reanude en la sentencia inmediata posterior a la que provocó el error.

RESUME <número de línea> hace que la ejecución se reanude en el número de línea especificado.

RETURN

Ver GOSUB.

STOP

La sentencia STOP se puede incluir en cualquier parte del programa para finalizar la ejecución y retornar al nivel de mandatos. Cuando el ordenador encuentra un STOP, visualiza el mensaje:

```
Break in xxxx
```

(donde xxxx es el número de la línea que contiene la sentencia STOP) en la pantalla. Se puede reanudar la ejecución mediante el comando CONT. Vd. mismo puede comprobar esto añadiendo:

```
75 STOP
```

al programa que vimos con la sentencia RESTORE. A diferencia de la sentencia END, la sentencia STOP no cierra los ficheros.

SWAP <variable>, <variable>

SWAP intercambia los valores de dos variables, así si A = 10 y B = 5, con SWAP A,B hacemos A = 5 y B = 10. La sentencia SWAP se puede aplicar a cualquier tipo de variable siempre y cuando éste sea el mismo para las dos variables.

Sumario

¡Enhorabuena! Ya ha terminado Vd. el capítulo, podríamos decir, más difícil del libro. Esperamos que no le haya resultado demasiado penoso. A lo largo de él, Vd. ha aprendido más de setenta mandatos y sentencias del MSX-BASIC, los más importantes, junto con las nociones fundamentales para encadenarlos en los programas. Esperamos haberle presentado los conceptos básicos de la programación de forma amena e interesante. Vd. debe ser capaz de utilizar esta sección del libro, como una guía de referencia cuando siga leyendo los capítulos restantes y vaya encontrándose con programas cada vez más avanzados.

En el siguiente capítulo, seguiremos progresando sobre las bases de lo que acaba de aprender, presentándole programas con estructuras más

complejas y describiendo los pocos mandatos que nos faltan por describir: sentencias y mandatos que no hemos considerado adecuado incluirlas en este capítulo. También veremos con mucho más detalle el tipo de los datos y cubriremos las funciones aritméticas más útiles contenidas en el MSX-BASIC.

Una vez que haya aprendido todo esto, Vd. tendrá ya un conocimiento de su ordenador y su lenguaje, suficiente como para poder escribir Vd. solo un programa que realice cualquier trabajo que a Vd. se le ocurra. En los siguientes capítulos, hemos incluido un número de esqueletos de programas para proporcionar la base de las aplicaciones de puntuaciones de exámenes, de juegos o aplicaciones musicales. Vd. puede adaptarlos en la forma que mejor sirva a sus intereses o a sus necesidades. En los dos últimos capítulos veremos el aspecto más esotérico de la programación utilizando los Macro-Lenguajes de la Música y los Gráficos.

Cómo trabajar con números

A estas alturas Vd. ya sabe perfectamente que los ordenadores precisan que se les proporcione algún tipo de información antes de poder hacer cualquier trabajo útil. En este capítulo veremos los tipos de datos que el ordenador puede manejar, y cuando debe emplearse cada uno de los tipos.

El MSX-BASIC soporta una serie de tipos diferentes de datos. En total hay siete tipos disponibles que son: entero, punto fijo, punto flotante, hexadecimal, octal, binario y carácter.

Los veremos todos, más adelante.

Los datos se pueden incorporar al programa en una de estas dos formas: como una constante o como una variable. Una constante no modifica su valor cuando el programa se ejecuta. Una variable puede modificarse durante la ejecución del programa, es decir, si Vd. quiere puede alterar su valor.

Constantes

En primer lugar, ¿cómo declarar constantes para cada uno de los diferentes tipos de datos? Bueno, vamos a presentarle cada uno de estos tipos y veremos cómo se hace esta declaración.

Un entero es sencillamente otra definición para un número completo, es decir, para aquellos números que no contienen el punto decimal. Deben estar incluidos en el rango de –32.768 a 32.767. Para incluirlos en los programas, sólo hay que escribir el número dentro de una sentencia MSX-BASIC. Por ejemplo, podemos sumar 10 a 5 con el siguiente programa:

```
10 PRINT 10 + 5
```

No importa el número de veces que Vd. ejecute este pequeño programa. el resultado final será siempre la visualización del valor 10 + 5 (15). Si forma parte de un programa, Vd. queda sujeto al valor de la constante, para bien o para mal.

Veamos ahora el tema de las constantes en **punto fijo**. Pueden ser números reales positivos o negativos y pueden contener una parte fraccionaria. Si existe la parte fraccionaria, se considera así sin tener que decir que tienen un punto decimal. Una constante en punto fijo queda declarada en un programa mediante la combinación de un sólo punto decimal y una serie de números.

–3.645, 123.87, 0.013, –0.56746 y .78 son valores aceptables para constantes de este tipo.

Otro tipo de constantes que puede contener un punto decimal son las de **punto flotante**. Son números positivos o negativos y se les puede declarar de las siguientes formas. Un método es la utilización de lo que se llama notación científica. Los números están representados por tres componentes: **la mantisa**, las letras E o D y el **exponente**. Esto queda explicado mejor con un ejemplo. El número 753000 se representaría como 7.53E5. La mantisa en este caso es 7.53 y el exponente es 5. Lo que está indicando la sentencia es que el número se puede expresar como 7.53 multiplicado por 10 elevado a 5, o dicho más sencillamente, es el equivalente del obtenido al mover el punto decimal 5 lugares a la derecha. Si el exponente es negativo, entonces el punto decimal se mueve hacia la izquierda. El exponente sólo puede ser un valor entero del rango –64 a 63.

La sustitución de D por E significa que el número debe almacenarse con una precisión de 14 cifras decimales en lugar de las 6 cuando se utiliza E. Los siguientes son ejemplos válidos de números en punto flotante:

Valor	Representación en punto flotante
12400	12.4E3
0.00683	6.83E-3
– 0.00004077032	– 4.077032D-5
534500210	5.3450021D8

Todos los tipos numéricos de datos se pueden almacenar con diferentes niveles de precisión. Los números en precisión sencilla (o simple) pueden tener hasta 6 dígitos decimales; los números en doble precisión pueden tener hasta 14 dígitos decimales. Vd. establece la precisión de un número utilizando E o D para constantes en punto flotante o los símbolos # o ! para los restantes tipos de números. Ya ha visto cómo declarar números en punto flotante con precisión simple o con doble precisión . Los números reales y los enteros, con precisión simple, se obtienen escribiendo un signo de exclamación al final de una serie de dígitos, por ejemplo 145! ó 76.6!.

Utilizando la letra D se consiguen números en punto flotante con doble precisión, y también escribiendo un símbolo # opcional al final de un número en punto fijo o de un entero. El # es opcional porque tan pronto como se conecta el ordenador MSX, el MSX-BASIC asume que todos los números tendrán doble precisión, a no ser que, claro está, Vd. le diga lo contrario al ordenador. La doble precisión es la precisión asumida por defecto en el MSX-BASIC. Así pues 5600 es un valor con doble precisión igual que 4573.56, 12.93 #, –89.9 # y 12314.

Quizás le resulten completamente nuevas las constantes **hexadecimales**. Los valores hexadecimales son números representados en base 16. En lugar de las unidades, decenas, centenas, etc. del sistema normal, el hexadecimal tiene unidades, grupos de 16, grupos de 256 (16 × 16) y así sucesivamente. Para dejar esto más claro, aquí vemos cómo se cuenta hasta 16 en hexadecimal.

Valor decimal :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor hexadecimal:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Observe que por encima del 9 y hasta el 16, los números se convierten en letras, en hexadecimal. Con el código hexadecimal (Hex en forma abreviada) se pueden tener números como "FFFF" o "BCDD". Los números hexadecimales se utilizan muy a menudo para programar en código máquina así que Vd. no debe preocuparse demasiado por ellos. Este es el único sitio del libro en el que se les hace referencia. Para declararlos, basta con escribir el prefiio &H delante del número. Ejemplos de números hexadecimales son: &H3FA, &H1650 y &H2FE.

Las constantes **octales** son otros vestigios que nos quedan de la época dorada del cálculo con los ordenadores. Así como el sistema hexadecimal utiliza el 16 como base de numeración y el decimal el 10, el sistema octal utiliza el 8. Todavía hay gente que utiliza los números octales, aunque posiblemente Vd. no quiera utilizarlos.

Sin embargo, a continuación damos la forma de contar hasta 16 en el sistema octal:

Valor decimal :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Octal :	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20

Y de nuevo se utiliza un prefijo para declarar una constante octal. En lugar de &H, para el hexadecimal, utilizamos &O. Ejemplos de constantes octales son: &O12, &O5643 y &O129.

Las constantes **binarias** vienen definidas por el prefijo &B. El sistema binario tiene el 2 como base de numeración, como dijimos en el Capítu-

lo 1. Utilizaremos estas constantes un poco más adelante. Como ya hemos explicado, la comunicación más elemental con el ordenador se basa en los 1's y 0's. Ejemplos de constantes binarias son: &B11110000, &B00101110 y &B11001101. En un sólo byte, Vd. puede representar cualquier número del 0 al 255.

El último tipo de datos disponible es el de **cadena de tipo carácter.** Una cadena de tipo carácter es cualquier cosa encerrada entre dobles comillas. Las letras de la A a la Z, tanto mayúsculas como minúsculas los números del 0 al 9 y cualesquiera otros elementos del juego de caracteres del MSX-BASIC son cadenas de tipo carácter si están escritas entre dobles comillas.

Las cadenas pueden tener una longitud de hasta 255 caracteres. He aquí algunos ejemplos:

"papa", "1234", "$% ^*", " ¡Soy una cadena!", "345.687".

Se utiliza muy frecuentemente en las sentencias PRINT, por ejemplo, para visualizar un mensaje en un programa:

```
10 PRINT "Hola, soy una cadena de tipo c
onstante"
20 PRINT "No puedo ser modificada mientr
as se esté ejecutando un programa"
```

Una vez descritas las constantes de tipo cadena, hemos cubierto todas las constantes disponibles. En la siguiente sección veremos las variables, cómo declararlas y cómo utilizarlas.

Variables

Variables son nombres que Vd. puede utilizar, dentro de un programa, para representar valores. Pueden ser modificadas durante la ejecución del programa. Al igual que las constantes, pueden ser de diferentes tipos, cuatro en total.

Empezaremos viendo cómo se definen las variables. Como con cualquier otro tema en programación, primero hay que aprender algunas reglas básicas. El nombre de la variable, puede ser de la longitud que Vd. quiera, suponiendo, claro está, que no va Vd. a agotar la memoria disponible. El nombre de una variable *debe* comenzar siempre con una letra.

No se permiten los espacios en el nombre de la variable. No se le puede dar a una variable, un nombre que sea una palabra reservada del BASIC o que la contenga. Dichas palabras quedan reservadas para el

MSX-BASIC y el lenguaje no tiene consideraciones con el programador que juega con ellas. Podría parecer que todas estas reglas son algo restrictivas, pero Vd. se habituará enseguida a ellas. Por ahora, no cometerá muchas equivocaciones utilizando solamente las letras de la A a la Z.

Se pueden declarar las variables de un tipo específico de datos mediante un carácter de declaración de variables que debe escribirse al final del nombre. Estos caracteres son:

% Variable entera
! Variable de simple precisión
Variable de doble precisión
$ Cadena de tipo carácter

A continuación damos una serie de nombres válidos y no válidos de variables:

Válido : Rango% A$ MSX! Trayectoria Nombre10$ PI%
No válido: 15X NAMELISTS$ 2NOMINA DIM%

Otro método alternativo para la declaración del tipo de las variables es la utilización de la sentencia DEF. El formato general de esta sentencia es:

DEF <tipo de variable> [<expresión>] , [<expresión>]

El tipo de variable puede ser declarado con **INT**, **SNG**, **DBL**, y **STR**, que son los especificadores para los tipos de datos enteros, de simple precisión, de doble precisión y de cadena respectivamente. La expresión que se incluye en la sentencia es una letra o un rango de letras. Por ejemplo, si se ejecuta la sentencia DEFINT A, todas las variables cuyo nombre empiece por la letra A serán de tipo entero. Si se ejecuta la sentencia DEFSTR A-Z, se asume que todas las variables del programa son de tipo cadena. Para cambiar el tipo de datos de una variable determinada después de la utilización de una sentencia DEF, hay que utilizar los caracteres de declaración de variables. El ejemplo que damos a continuación es ilustrativo sobre la sentencia DEF.

```
10 DECLARE A-Z:REM DECLARAR TODAS LAS VA
RIABLES COMO CADENAS DE CARACTERES
20 S="RAMON ":R="SANCHEZ"
30 PRINT S+R
```

```
40 PRINT A
50 A%=100
60 PRINT A%*10
```

Observe que las cadenas S y R no precisan del carácter de declaración $. Cuando se visualiza la variable A, en la línea 40, tenemos una cadena vacía, en la línea 50 se convierte en una variable entera.

El último tipo de variables es el array subindicado que ya hemos visto. Estudiaremos los arrays con más detalle en los capítulos posteriores.

Cuando haya decidido el nombre que va a dar a sus variables, puede incluirlas en los programas y empezar a darles valores. Al proceso de dar un valor a una variable se le conoce con el nombre de **asignación de variables**. Pruebe a ejecutar el siguiente programa:

```
10 REM Asignación de variables
20 PRINT NUMERO%
30 INPUT NUMERO%
40 LET NUMERO%=NUMERO%+1
50 PRINT NUMERO%
```

La única variable de este programa es NUMERO% una variable entera. Cuando se visualiza su valor, en la línea 20, NUMERO% es 0. Esto es debido a que cuando el MSX-BASIC se encuentra, en el programa, con un nombre de variable nuevo asume, inicialmente, que su valor es 0. En la línea 30 tenemos una sentencia INPUT en la que se le solicita que asigne un valor a NUMERO%. En la línea 40, la sentencia LET suma 1 a este valor y el resultado se visualiza en la línea 50.

Si como respuesta al mensaje de diálogo ? intenta Vd. introducir una letra o una palabra, el BASIC le sacará el mensaje –'?redo from start'. El motivo es que (como recordará) Vd. le ha dicho al ordenador que lo que debe esperar a continuación es el tecleo de un valor entero. ¡Y quedará a la espera hasta que Vd. introduzca un entero! ¡Cuánta paciencia tienen los ordenadores...!

Seguro que Vd. recuerda también que la palabra LET de la línea 30 es opcional. La sentencia NUMERO% = NUMERO% + 1 hubiese hecho exactamente lo mismo. De esta forma se pueden asignar valores a cualquier variable.

Ahora intente cambiar los signos % por signos $, convirtiendo a NUMERO en una variable de tipo cadena. Si ejecuta el programa ahora, se obtendría otro mensaje del MSX-BASIC. Esta vez vendrá acompañado de un pitido (beep) y no es tan amable. De hecho, Vd. se encontrará

con el mensaje de error "Type mismatch in line 30" (música dramática, por favor). Ya entrará en contacto con muchos de esos mensajes pequeños e irritantes cuando esté programando —en el Apéndice A se incluye una lista completa de todos los mensajes de error. La razón por la cual se visualizó ese mensaje era que el ordenador estaba intentando sumar 1 a la palabra que Vd. hubiese tecleado y se quedaba hecho un verdadero lío. Es como si sumásemos peras y manzanas —son dos clases distintas de objetos y no se las puede tratar de la misma forma. Si Vd. quita la línea errónea (40) entonces todo va bien: se visualizará cualquier palabra que Vd. teclee, y el ordenador no tendrá que intentar sumar "peras y manzanas". Pero antes de seguir adelante intente esto: Cambie la línea 40 a:

```
LET NUMEROS = NUMERO$ + "1"
```

Al poner el 1 entre comillas, se ha convertido en una cadena y el ordenador se encuentra feliz sumando cadenas.

Resulta muy útil conocer la cantidad de memoria que el ordenador precisa para almacenar los valores de las variables. La tabla que damos a continuación muestra la cantidad de espacio que se necesitará para cada tipo de variable, incluyendo las variables de arrays.

VARIABLES

Tipo de Datos	Número de bytes
Entero	2
Precisión simple	4
Precisión doble	8
Cadenas	1 por carácter + 3 ej. la cadena "PEPE" necesitaría un total de 7 bytes

ARRAYS

Tipo de Datos	Número de Bytes
Entero	2 por elemento
Precisión simple	4
Precisión doble	8

Habiendo presentado todos los diferentes tipos, vamos a ver el aspecto de la programación de todas estas cosas y a poner en escena algunas nociones matemáticas muy sencillas.

Expresiones Matemáticas

Las matemáticas no es la más popular de las asignaturas. Desgraciadamente, los ordenadores son francamente buenos en matemáticas, así que no podemos pasarlas por alto. Para empezar con lo más sencillo, teclee:

```
PRINT 10*10
```

El ordenador le contestará con '100'.

El ordenador se está comportando como una simple calculadora. Lo que Vd. le está pidiendo es que visualice el resultado de multiplicar 10 por 10. Como no hay disponible un signo para la multiplicación (x), el ordenador utiliza un asterisco. Intente también con las tres siguientes:

```
PRINT 10 + 10
PRINT 10 – 5
PRINT 10/2
```

Eran operaciones de suma, resta y división –todo lo que Vd. necesita para las funciones aritméticas sencillas. Pongámoslas en los programas. El siguiente programa realiza únicamente las cuatro operaciones aritméticas de los dos números que Vd. le proporcione:

```
10 INPUT "Dos números, por favor";A,B
20 PRINT "A + B = ";A+B
30 PRINT "A - B = ";A-B
40 PRINT "A * B = ";A*B
50 PRINT "A / B = ";A/B
```

No es un programa asombrosamente complejo pero ¡funciona! Sigamos ahora con un programa más ambicioso. El objetivo de este programa es aceptar del usuario palabras y números y ejecutar alguna operación aritmética simple. Se debe introducir el nombre de la operación matemática, en mayúsculas, y dos números. Por ejemplo, un usuario podría escribir: MULTIPLICAR, 5,4 y el resultado sería 20. Antes de que Vd. pueda hacer esto, debe aprender unas cuantas cosas nuevas.

El programa "sabe" que hay cuatro palabras asociadas con las operaciones aritméticas. Están representadas como constantes, en el programa. Lo que el ordenador hace es comparar la palabra escrita por el usuario con sus propias palabras constantes. El MSX-BASIC compara dos

cadenas de caracteres utilizando el signo igual, exactamente igual que si tuviera que comparar dos números.

Así pues la sentencia que el programa utiliza para decidir si debe o no multiplicar, dividir, sumar o restar es la sentencia IF que ya vimos anteriormente. Si se teclea la palabra SALIDA en lugar del nombre de una operación matemática, el programa visualiza un amable mensaje de despedida y termina. En caso contrario, compara la palabra introducida con los cuatro términos artiméticos. Si no existe coincidencia con ninguno de ellos, el ordenador le dice al usuario que no tiene ni idea de qué hacer y le pide que escriba la información de nuevo. Este programa sería algo así:

```
10 REM CALCULADORA SENCILLA
20 REM PAGINA DE INSTRUCCIONES
30 CLS
40 PRINT "PROGRAMA CALCULADOR":PRINT "---
-------------------"
50 PRINT "Teclee el nombre de la ope
ración "
60 PRINT "matemática seguida de dos
números.":PRINT
70 PRINT "La operación y los dos números
 deben ";
80 PRINT "estar separados por comas, com
o por ejemplo:"
90 PRINT "MULTIPLICAR,5,4"
100 PRINT "Teclee SALIDA,0,0 cuando haya
terminado.":PRINT
110 PRINT:INPUT "Entre los datos :";A$,X
,Y
120 IF A$="SALIDA" THEN GOTO 180
130 IF A$="SUMAR" THEN SUM=X+Y:PRINT "La
suma de ";X;" e ";Y;" es ";SUM:GOTO 110
140 IF A$="RESTAR" THEN SUM=X-Y:PRINT
"El resultado de ";X;" menos ";Y;" es ";
SUM:GOTO 110
150 IF A$="MULTIPLICAR" THEN SUM=X*Y:
PRINT "El producto de ";X;" por ";Y;" es
";SUM:GOTO 110
160 IF A$="DIVIDIR" THEN SUM=X/Y:PRINT "
La división de ";X;" entre ";Y;" es";SUM
```

```
:GOTO 110
170 PRINT "Lo siento, no comprendo.";A$;
" Pruebe otra vez":GOTO 110
180 PRINT "Hasta la próxima!"
190 END
```

Observe la utilización de las sentencias REM. Son muy útiles cuando Vd. deja un programa grabado en cinta durante mucho tiempo y luego no puede recordar qué es lo que hace. También le permiten seguir la lógica (¡o lo que Vd. creía lógica!) del programa, si en fechas posteriores desea modificarlo. Siga el programa a través de ellas —es relativamente fácil de comprender.

La suma, resta, multiplicación y división no son las únicas operaciones matemáticas posibles con el MSX-BASIC. También están disponibles **la potencia**, **la negación**, **la división entera** y el **módulo aritmético**. Todos los operadores matemáticos obedecen lo que se conoce como **reglas de prioridad**. Cuando el intérprete del MSX-BASIC ve una línea llena de estos operadores aritméticos, necesita decidir sobre cuál será la operación que realizará en primer lugar. Dada la sentencia:

```
A = B + C*D
```

resulta difícil para el MSX-BASIC decidir si lo que Vd. quiere es sumar B y C primero y el resultado multiplicarlo por D o multiplicar C por D y añadir el producto a B. Si sustituye las letras de la ecuación anterior por números, comprenderá mejor estos problemas.

Suponiendo que B es igual a 2, C igual a 4 y D igual a 6, para el primer caso, sumar B a C y después multiplicar por D, los pasos serán algo parecido a:

Primero sumar 2 y 4 = 6
Multiplicar 6 × 6
Dando A = 36

Y para el segundo caso, en el que se ejecuta primero el producto C*D, sumándose el resultado a B:

Primero multiplicar 4 por 6 = 24
Sumar 24 a 2
Dando A = 26

Como Vd. puede ver, se obtienen dos valores totalmente diferentes para lo que parece una sentencia sencilla. De hecho, el MSX-BASIC

obtendría el número 26. Esto es debido a que se le ha indicado que todas las multiplicaciones deben realizarse antes de las sumas. De aquí el término "prioridad".

La multiplicación tiene prioridad sobre la suma. El orden de estas prioridades para todos los operadores aritméticos queda sumarizada a continuación:

Símbolo	Operación	Ejemplo
^	Potencia	A^B
–	Negación	– A
*, /	Multiplicación y división en punto flotante	A*B A/B
¥	División entera	A¥B
MOD	Módulo aritmético	A MOD B
+, –	Suma y resta	A + B A – B

Se puede alterar este orden mediante la utilización de los paréntesis. Siguiendo con el ejemplo anterior, si queremos ejecutar la suma antes que la multiplicación, deberíamos escribir la expresión de la siguiente forma:

```
A = (B + C)*D
```

La operación escrita entre paréntesis se evalúa primero. Si B es igual a 2, C es igual a 4 y D es igual a 6 el resultado sería 36.

La **exponenciación** o **potencia** es simplemente el proceso de elevación de un número a una potencia. Por ejemplo, si Vd. escribe:

```
PRINT 10^2
```

el resultado que se visualizará es 100. En otras palabras, la sentencia que Vd. ha tecleado significa "elevar 10 a la segunda potencia". La notación utilizada por el MSX-BASIC en este caso es equivalente a lo siguiente: 102. A continuación damos otros ejemplos y sus resultados:

Sentencia	Resultado	Notación Equivalente
PRINT 10^4	10000	104
PRINT 2^8	256	28
PRINT 90^2.5	76843.347142016	902.5
PRINT 1.37^40	294321.9730757	1.3740

La **división entera** se diferencia de la división normal en punto flotante en lo siguiente. Los operandos quedan truncados a valores enteros, se efectúa la división y el resultado se trunca dando un entero. Truncar un número significa "eliminar" todo lo que venga después del punto decimal. Una vez truncado, 2.57 se convertiría en 2, 9.999999 se convertiría en 9 etc. Quizás no le resulte familiar el símbolo utilizado para la división entera. Es el equivalente japonés del signo monetario y normalmente representa el YEN, la moneda del Japón. Si rompemos el proceso de la división entera, en pasos, Vd. verá exactamente cómo funciona. Pongamos un ejemplo. Dividamos 345.978 entre 12.866 con división entera. A continuación vemos los pasos utilizados por el BASIC:

```
Sentencia: PRINT 345.978¥12.866
           1 Truncar   :  345.978 a 345
           2 Truncar   :   12.866 a 12
           3 Dividir   :      345 entre 12
           4 Resultado =   28.75
           5 Truncar   :   28.74 a 28
           6 Resultado =   28
```

Si hubiésemos utilizado la división normal (con el símbolo /) el resultado final habría sido bastante diferente (26.890875174879). Por último, vamos con el **módulo aritmético**. Esta operación viene indicada por el operador MOD. Lo que hace es dar el resto entero de una división. Si escribe 'PRINT 7.86 MOD 2' el resultado sería 1. La forma en que el MSX-BASIC llega a este resultado es truncando 7.86 a 7, dividiendo 7 entre 2 con lo cual queda el resto 1. De nuevo damos una lista para que vea algunos ejemplos:

Sentencia	Resultado
1234.4321 MOD 11.31	2 (1234/11 = 112 con resto 2)
100 MOD 10	0 (100/10 = 10 con resto 0)
76 MOD 13	11 (76/13 = 5 con resto 11)
99,99 MOD 31	6 (99/31 = 3 con resto 6)

Con cualquier tipo de división hay un punto importante que debe Vd. tener en mente: *¡No divida nada por cero!*. El ordenador generará siempre un mensaje de error si Vd. intenta dividir cualquier número por 0. Debe ser especialmente cuidadoso con el operador MOD y la división entera. Si Vd. tiene una sentencia del tipo 7 MOD 0.45 ó 7 ¥ 0.45, ambos operadores truncarán 0.45 a 0 produciéndose el mensaje de error. Preste atención también a estas operaciones cuando utilice variables. Vd. puede intentar dividir un número utilizando una variable a la que no se le ha asignado ningún valor.

Aquí terminamos con los operadores aritméticos (así que puede respirar tranquilo), pero estos no son el único tipo de operadores disponibles. En la próxima sección veremos dos grupos más. Antes de que caiga en la desesperación le diremos que son de gran utilidad y sobre todo muy fáciles de comprender.

Operadores Lógicos y Relacionales

Estos tipos de operadores se utilizarán frecuentemente en los programas largos y probablemente Vd. utilice los operadores relacionales incluso más que los operadores aritméticos, por lo tanto es conveniente que empecemos a conocerlos.

Todo lo que los **operadores relacionales** hacen es comparar cosas. Tan simple como eso. Pueden ver si dos cosas son iguales, si una es menor o mayor que la otra, o si son completamente distintas.

Una nueva tabla es suficiente para mostrar todos los operadores relacionales. Son *muy* sencillos de utilizar. Son seis, y aquí están junto con la descripción de lo que comparan:

Operador	Función	Ejemplo
=	Comprueba si una cosa es igual a otra	X = Y
<>	Comprueba la desigualdad	X <> Y
<	Menor que	X < Y
>	Mayor que	X > Y
<=	Menor o igual que	X <= Y
>=	Mayor o igual que	X >= Y

Estos operadores son especialmente útiles en las sentencias IF... THEN. Por ejemplo, si tenemos una serie de notas de exámenes de una clase probablemente nos gustaría conocer a qué grupo quedaría asignado cada estudiante. Pondremos esto en práctica con el siguiente programa. Se introducen en el programa los nombres de los estudiantes, sus asignaturas y sus notas de examen de cada una de las tres asignaturas. El programa verifica o **valida** los datos de entrada sobre la marcha de manera que es imposible introducir notas como –34% ó 230%. Las notas correctas variarán de 0 a 100. La salida final del programa es el nombre del estudiante, el grupo asignado para cada examen y el grupo para la media de las tres asignaturas. Después de procesar la información de cada estudiante el programa pregunta si se le va a proporcionar más información.

Incluimos una nueva función: INT. INT redondea por defecto, o trunca un número. (Vea en el Apéndice A la lista completa de todas las funciones del MSX-BASIC). También utilizaremos una subrutina para validar los datos numéricos. Como hay que verificar las notas de tres exámenes distintos utilizando la misma subrutina, antes de saltar a la subrutina, se asigna el valor de cada nota a una variable llamada TEMP!. Cada vez que se comprueba la nota de una asignatura, cambia el valor de TEMP!. Utilizaremos la variable TEMP! de la misma forma para asignar los grupos a las notas, mediante otra subrutina.

Se pueden sumarizar los pasos del programa en palabras. Este sistema de planificación de un programa se conoce como un **algoritmo** (esencialmente es lo mismo que el organigrama, del que ya hemos hablado). Este es el algoritmo del programa:

EMPEZAR Introducir el nombre del estudiante
Introducir notas de Biología, Química y Física
Comprobar que los datos sean correctos
Convertir cada nota en su grupo correspondiente
Media de cada una de las notas y redondear por defecto el resultado
Visualizar el nombre, los grupos de Biología, Química y Física más la media de los exámenes
Preguntar si hay que procesar más datos en cuyo caso ir a EMPEZAR de nuevo, en caso contrario, parar.

Veamos esto en el organigrama de la figura 6. A continuación se lista el programa final. Está muy bien comentado para mostrarle como se ha traducido cada paso del algoritmo al MSX-BASIC.

```
10 REM PROG. DE NOTAS DE LOS EXAMENES
20 REM LIMPIAR PANTALLA E INTR. DATOS
30 CLS
40 PRINT "Teclee el nombre del estudiant
e ":INPUT S$
50 PRINT "Teclee la nota de Biología ":I
NPUT BIOL!
60 TEMP!=BIOL!:GOSUB 370:BIOL!=TEMP!:REM
COMPROBACION DE NOTAS
70 PRINT "Teclee la nota de Química ":IN
PUT QUIM!
80 TEMP!=QUIM!:GOSUB 370:QUIM!=TEMP!:REM
COMPROBACION DE NOTAS
```

```
90 PRINT "Teclee la nota de Física ";INP
UT FISI!
100 TEMP!=FISI!:GOSUB 370:FISI!=TEMP!:RE
M COMPROBACION DE NOTAS
110 REM ASIGNACION DE LOS GRUPOS
120 TEMP!=BIOL!:GOSUB 430:G1$=GRUPO$
130 TEMP!=QUIM!:GOSUB 430:G2$=GRUPO$
140 TEMP!=FISI!:GOSUB 430:G3$=GRUPO$
150 REM CALCULO DE LA MEDIA PARA LOS TRE
S CURSOS
160 MED%=INT((BIOL!+QUIM!+FISI!)/3)
170 REM VISUALIZAR TODOS LOS DATOS
180 CLS: PRINT "RESULTADOS DEL ESTUDIANT
E":PRINT "------------------------":PRI
NT
190 PRINT "NOMBRE DEL ESTUDIANTE : ";S$
200 PRINT "GRUPOS:":PRINT
210 PRINT "BIOLOGIA : ";G1$:PRINT "QUIMI
CA  : ";G2$:PRINT "FISICA   : ";G3$
220 PRINT
230 PRINT "MEDIA DE LOS EXAMENES : ";MED
%;"%"
240 PRINT:PRINT
250 REM VER SI EL USUARIO DESEA SEGUIR O
NO
260 PRINT "DATOS DE MAS ESTUDIANTES" : I
NPUT "PARA PROCESARLOS (S O N)";RESP$
270 REM SI EL USUARIO TECLEA "S" ENTONCE
S SEGUIR SI NO TERMINAR
280 IF RESP$="S" THEN GOTO 30
290 IF RESP$="N" THEN GOTO 330
300 REM RESPUESTA NO VALIDA - INTENTELO
DE NUEVO
310 PRINT "POR FAVOR CONTESTE CON UNA 'S
'":PRINT "O 'N'":GOTO 260
320 REM TRATAMIENTO DE LA N - VISUALIZAR
UN MENSAJE DE DESPEDIDA
330 PRINT "FIN DE ENTRADA DE DATOS"
340 END
350 REM*****SUBRUTINAS****
```

```
360 REM*SUBRUTINA PARA VERIFICAR NOTAS
370 IF TEMP!<=100 AND TEMP!>=0 THEN RE
TURN:REM LOS DATOS SON CORRECTOS
380 REM PERMITIR AL USUARIO LA CORRECCIO
N DE ERRORES
390 PRINT "ESTE VALOR NO ESTA COMPRENDID
O " : PRINT "EN EL RANGO PERMITIDO PARA
LAS NOTAS"
400 PRINT "INTENTE CON UN VALOR BUENO ":
INPUT TEMP!
410 GOTO 370
420 REM* SUBRUTINA -- OBTENCION DE GRUPOS
430 IF TEMP!>=65 THEN GRUPO$="A":RETURN
440 IF TEMP!>=55 THEN GRUPO$="B":RETURN
450 IF TEMP!>=45 THEN GRUPO$="C":RETURN
460 IF TEMP!>=35 THEN GRUPO$="D":RETURN
470 IF TEMP!>=25 THEN GRUPO$="E":RETURN
480 IF TEMP!<25 THEN GRUPO$="MAL":RETUR
N
```

Los operadores relacionales se utilizan para tres cosas fundamentales en este programa: para comprobar que los números introducidos son válidos, para decidir sobre la de asignación de los grupos y por último para analizar la respuesta que el usuario da a la pregunta "¿DATOS DE MAS ESTUDIANTES?". En el caso de la asignación de los grupos, todo lo que el programa hace es efectuar algunas preguntas. Si la nota del examen es MAYOR QUE o IGUAL QUE (GREATER THAN OR EQUAL TO) el grupo correspondiente a 65, que es la frontera del grupo A, entonces se asigna "A" a la variable GRUPO$, y así con todos los grupos disponibles.

Vd. habrá observado algo extraño en la sentencia con la que se comprueba si los números introducidos están dentro del rango permitido —línea 370. Esa línea contiene la palabra AND que es un **operador lógico**. Este es el último tipo de operadores que discutiremos en el MSX-BASIC.

Los operadores lógicos se utilizan sobre todo en las operaciones IF...THEN. Realizan las operaciones lógicas o Booleanas. Son NOT, AND, OR, XOR y EQV. Veamos la condición AND que hemos utilizado en el programa anterior:

```
370 IF TEMP! <= 100 AND TEMP! >= 0 THEN RETURN
```

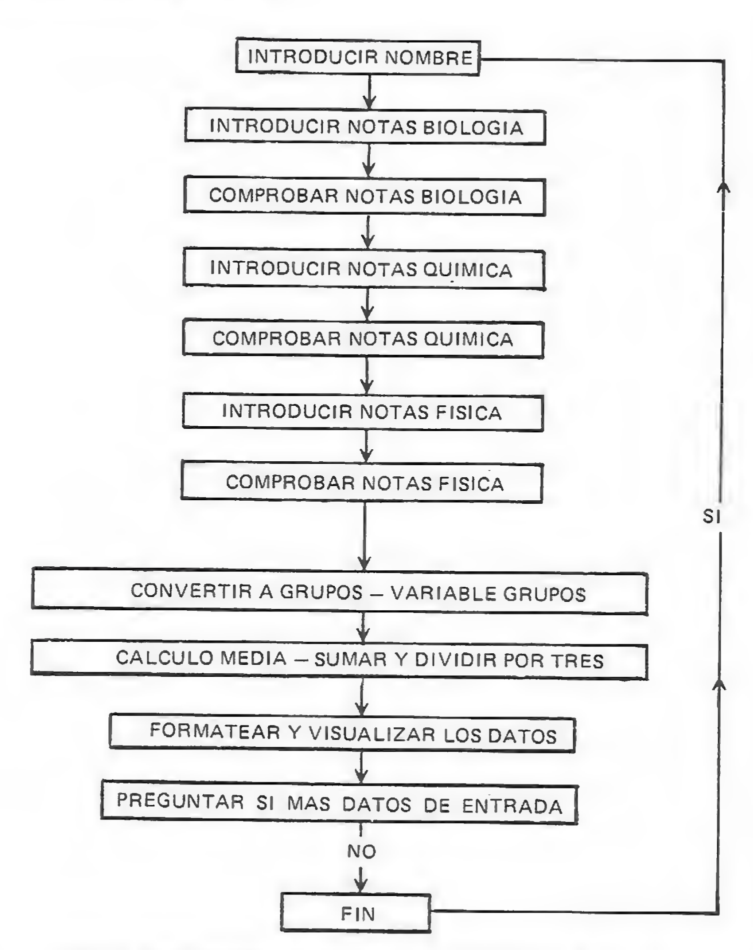


Fig. 6.—Organigrama para el programa de cálculo de grupos.

Lo que esta línea está comprobando es que ambas condiciones sean verdaderas. Si ambas condiciones fuesen ciertas THEN (ENTONCES) se tomaría una cierta acción, en este caso el retorno de la subrutina.

En la lógica del ordenador "verdadero" (true) significa 'sí' y "falso" (false) significa 'no'. Como ejemplo, considere la variable X, donde X es igual a 10. Las siguientes expresiones reflejan la forma en que se aplican los conceptos verdadero y falso a las operaciones lógicas:

X < 20 Verdadero
X = 10 Verdadero
X > 10 Falso

Podemos ampliar esto con la variable Y, donde Y = 50. Veremos ahora cómo se evalúa una expresión como esta:

(X = 10) AND (Y = 40)

En este caso la respuesta sería 'falso' porque no se cumple una de las condiciones. Para que fuese verdadero, ambas condiciones deberían ser verdaderas. Por el contrario, una expresión como:

IF X = 10 OR Y = 40

dará una respuesta 'verdadero' porque una de las dos expresiones es verdadera.

Una de las razones por las que parece que el ordenador "piensa" es la comprobación condicional. Puede comparar dos valores y dar la impresión de que toma una decisión acerca de lo que debe hacer basado en su comparación. Todo esto utiliza la capacidad del ordenador para evaluar condiciones y expresiones matemáticas simples.

Vd. puede representar la forma en que trabaja el operador lógico utilizando una **tabla de verdad**. En este tipo de tabla, se representan los dos estados de las condiciones que se están comprobando como T (TRUE o 1) si la condición es verdadera, y F (FALSE ó 0) si es falsa. La salida de la operación lógica dependerá del estado en que se encuentren las dos condiciones. La tabla que damos a continuación sumariza los operadores lógicos. Se han listado en orden de prioridad.

NOT es TRUE sólo cuando la condición es falsa. Será FALSE cuando la condición sea verdadera. Por ejemplo:

```
10 X%=0
20 X%=NOT(X%)
```

X% se convierte en –1. Veamos otra utilización:

```
10 X%=-1
20 IF NOT X% THEN BEEP
```

El programa generará un pitido ya que NOT X% es FALSE (cero).

La tabla de verdad para el operador NOT es:

X	NOT X
T	F
F	T

AND es TRUE sólo cuando ambas condiciones sean verdaderas. Por ejemplo:

```
10 X=50:Y=100
20 IF (X=50) AND (Y=100) THEN BEEP
```

Como ambas condiciones son verdaderas, se generará un pitido.

La tabla de verdad para el AND es:

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

OR es TRUE cuando una o ambas condiciones son verdaderas, por ejemplo:

```
10 X=10:Y=40
20 IF (X<100) OR (Y>40) THEN BEEP
```

La comprobación da verdadera porque el valor de X cumple una de las condiciones.

La tabla de verdad para el OR es:

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

XOR (OR exclusiva) es TRUE solamente cuando una condición es verdadera y la otra falsa, por ejemplo:

```
10 X=10:Y=40
20 IF (X<100) XOR (X>40) THEN BEEP
```

El resultado es verdadero. Si Y fuese 100, la condición habría fallado.

La tabla de verdad para el XOR es:

X	Y	X XOR Y
T	T	F
T	F	T
F	T	T
F	F	F

EQV es TRUE cuando ambas condiciones tienen el mismo estado; es decir, ambas son falsas o ambas son verdaderas.

Ejemplos en los que la sentencia condicional es verdadera son:

```
10 X=10:Y=10
20 IF X=10 EQV Y=10 THEN BEEP
```

o

```
10 X=10:Y=10
20 IF X<>10 EQV Y<>10 THEN BEEP
```

La tabla de verdad para el EQV es:

X	Y	X EQV Y
T	T	T
T	F	F
F	T	F
F	F	T

IMP (Implicación). Extraño operador éste. La condición es FALSE solamente si la primera condición comprobada es verdadera y la segunda falsa. Por ejemplo:

```
10 X=10:Y=50
20 IF X=10 IMP Y=10 THEN BEEP
```

El programa anterior no emitirá el pitido ya que X = 10 es verdadero e Y = 10 es falsa.

La tabla verdad para el IMP es:

X	Y	X IMP Y
T	T	T
T	F	F
F	T	F
F	F	T

Con esto completamos nuestro recorrido, con guía, a través de los números y los operadores con los que puede trabajar el MSX-BASIC. A partir de ahora haremos un poco más de programación.

Utilización de Arrays

En esta sección estudiaremos la estructura de datos **array** con mucho más detalle, dando ejemplos de su utilización. Como Vd. ya sabe, los arrays se definen mediante la sentencia DIM. A través de esta sentencia DIM, Vd. le da un nombre al array así como sus dimensiones, es decir, el número de elementos que podrá contener.

El nombre del array también va a definir el tipo de datos de los elementos que se almacenarán en el array. Por ejemplo, si un array se define como:

DIM A$ (5)

se podrán almacenar en él hasta un máximo de 5 elementos siendo estos de tipo carácter. Vd. no puede mezclar los tipos de datos en un array. El único tipo de datos permitido en un array se conoce como **tipo base** del array.

A cada elemento del array se le asigna un único número, o se le **indexa**, para facilitar su referencia. El número utilizado para indexar un array se denomina **índice** (o **subíndice**) del array. El siguiente programa ayuda a recordar el color referenciado por un número determinado en el MSX-BASIC.

```
10 DIM A$(16)
20 FOR I=0 TO 15
30 READ CL$
40 A$(I)=CL$
50 NEXT I
60 CLS
70 INPUT "NUMERO DE COLOR, POR FAVOR ";N
%
```

```
80 IF N%<0 OR N%>15 THEN BEEP:GOTO 60
90 PRINT
100 PRINT "COLOR ";N%;" ES ";A$(N%)
110 REM INFORMACION DE LOS COLORES
120 DATA TRANSPARENTE,NEGRO,VERDE MEDIO
130 DATA VERDE CLARO, AZUL OSCURO
140 DATA AZUL CLARO,ROJO OSCURO, AZUL
GRISACEO
150 DATA ROJO MEDIO,ROJO CLARO,AMARILLO
OSCURO
160 DATA AMARILLO CLARO, VERDE OSCURO,
MAGENTA,GRIS,BLANCO
170 END
```

En la tabla siguiente tenemos una imagen del array A$. Cuando Vd. teclea un número, se comprueba primero que esté en el rango de números válidos. Si es correcto, el número tecleado sirve para "buscar" el nombre del color en el array A$.

Indice del Array	Contenido
0	"TRANSPARENTE"
1	"NEGRO"
2	"VERDE MEDIO"
3	"VERDE CLARO"
4	"AZUL OSCURO"
5	"AZUL CLARO"
6	"ROJO OSCURO"
7	"AZUL GRISACEO"
8	"ROJO MEDIO"
9	"ROJO CLARO"
10	"AMARILLO OSCURO"
11	"AMARILLO CLARO"
12	"VERDE OSCURO"
13	"MAGENTA"
14	"GRIS"
15	"BLANCO"

Podemos considerar al array como una tabla. La ventaja de utilizar los arrays radica en su rapidez. Intente escribir el programa anterior utilizando sentencias IF...THEN y verá lo que queremos decir:

```
80 IF N%=0 THEN PRINT "EL COLOR 1 ES TRA
NSPARENTE"
90 IF N%=1 THEN PRINT "EL COLOR 2 ES NEG
RO"
```

etc.

El array que hemos utilizado en el ejemplo anterior era un array unidimensional. Vd. puede tener también arrays de dos o incluso tres dimensiones. El siguiente programa utiliza un array bidimensional para almacenar tres grupos de estudiantes para tres asignaturas, como vimos en el anterior programa de notas de exámenes. El array ST está dimensionado de 3 por 3 elementos (Las cifras que aparecen entre paréntesis representan el número de estudiantes y el grupo).

Biología	Química	Física
ST(1,1)	ST(2,1)	ST(3,1)
ST(2,1)	ST(2,2)	ST(2,3)
ST(3,1)	ST(3,2)	ST(3,3)

Se utilizan dos arrays independientes, N$ y SB$ para almacenar los nombres de los estudiantes y los nombres de las asignaturas. El programa almacena los grupos y los nombres de los estudiantes y visualiza todos los datos en pantalla en forma de tabla.

```
10 DIM ST(3,3),N$(3),SB$(3)
20 FOR N=1 TO 3
30 READ A$
40 SB$(N)=A$:REM NOMBRE DE LA ASIGNATURA
50 NEXT N
60 DATA BIOLOGIA,QUIMICA,FISICA
70 FOR I=1 TO 3:REM ENTRADA DE DATOS
80 CLS
90 INPUT "NOMBRE DEL ESTUDIANTE ";S$
100 N$(I)=S$
110 FOR J=1 TO 3
120 PRINT SB$(J);" NOTA ";:INPUT SC
130 ST(I,J)=SC
140 NEXT J
```

```
150 NEXT I
160 CLS
170 REM SALIDA DE LA INFORMACION EN FORM
A DE TABLA
180 FOR I=1 TO 3
190 PRINT "NOMBRE ";N$(I)
200 PRINT
210 FOR J=1 TO 3
220 PRINT SB$(J),ST(I,J)
230 NEXT J
240 PRINT
250 NEXT I
260 END
```

Cómo ordenar las cosas

El aburrido proceso de clasificación de los datos es una de esas tareas pesadas que pueden ser realizadas por el ordenador con rapidez (¡y paciencia!). Durante muchos años, matemáticos y expertos en ordenadores han estado investigando sobre el tema de la clasificación. A continuación le presentamos una rutina de clasificación muy sencilla sin adentrarnos en la teoría. En estos ejemplos, utilizaremos arrays para almacenar los datos que vamos a clasificar.

Existe un sistema muy conocido de clasificación llamado **método de la burbuja**. Primero daremos el programa para la clasificación; después lo comentaremos.

```
10 CLS
20 INPUT "NUMERO DE ELEMENTOS A CLASIFIC
AR";N
30 DIM ARRAY(N)
40 FOR I=1 TO N
50 INPUT "NUMERO POR FAVOR ";X
60 ARRAY(I)=X
70 NEXT I
80 CLS
90 PRINT "SECUENCIA ORIGINAL DE NUMEROS"
:PRINT
100 FOR I=1 TO N:PRINT ARRAY(I):NEXT I
110 REM CLASIFICACION DE LOS DATOS
```

```
120 FOR I=2 TO N
130 FOR J=N TO I STEP -1
140 IF ARRAY(J-1)>ARRAY(J) THEN SWAP
ARRAY(J-1),ARRAY(J)
150 NEXT J
160 NEXT I
170 REM SALIDA DE DATOS CLASIFICADOS
180 PRINT:PRINT "DATOS CLASIFIC.":PRINT
190 FOR I=1 TO N:PRINT ARRAY(I):NEXT I
200 END
```

Esta versión de clasificación por el método de la burbuja clasifica los datos en orden ascendente. Para clasificarlos en orden descendente, reemplazar el signo mayor que (>) por el signo menor que (<).

El método de la burbuja analiza los elementos desde el último elemento 'más alto' del array hasta el segundo valor, comparando el valor actual con el inmediato siguiente. Si el valor del elemento siguiente al actual es más alto que el valor actual, entonces los dos valores intercambian sus posiciones. Cuando se ha realizado este análisis una vez, se tiene como primer elemento del array el de valor más bajo que queda descartado del proceso que se realiza empezando otra vez desde el elemento final del array. Suponga que tenemos los siguientes datos desclasificados:

100 7 9 1 514

Los pasos intermedios del proceso de clasificación aparecen en la siguiente tabla:

Valor de I	Valor de J	Orden de los Datos
2	5	1 100 7 9 514
3	5	1 7 100 9 514
4	5	1 7 9 100 514
5	5	1 7 9 100 514

Ya ve Vd. cómo el valor 100 "modifica" su posición en la secuencia de números. El método de la burbuja es un tipo de clasificación por intercambio, muy sencillo porque los valores del array sólo necesitan intercambiarse entre sí para colocarse en la posición correcta. También se puede utilizar esta rutina de clasificación para clasificar nombres, direcciones o cualquier clase de cadenas de caracteres —basta con cambiar los tipos de datos de los arrays utilizados así como las variables correspondientes.

Sin embargo esta clasificación tiene sus inconvenientes. Suponga que tiene una lista de elementos como 1 2 3 5 4. Esta lista sólo precisa un análisis para quedar en orden —intercambiando las posiciones del 4 y el 5. Pero el programa seguirá analizando todos los datos, incluso aunque ya estén clasificados. Este método no es adecuado cuando los números están *casi* en secuencia.

Bien, ¿y cómo le decimos al ordenador que ya está todo clasificado? Podemos decir que todo está en la secuencia correcta cuando no quedan valores que precisen intercambiar sus valores. Lo que se puede hacer es cambiar una variable de 0 a 1 si tienen lugar algún intercambio durante el análisis que el programa hace de la lista de números. Si después de una pasada, el valor de la variable es 1, el programa sabe que debe seguir tamizando los números hasta que estén en orden. A la variable utilizada en este caso se le conoce normalmente con el nombre de **flag** —indica que se ha producido un suceso. El programa comprobaría el valor del flag para ver si se ha producido algún intercambio.

Para que el programa anterior sea más eficiente, añada las siguientes líneas:

```
115 F = 0
155 IF F = 0 THEN GOTO 170
```

y modifique la línea 140 en este sentido:

```
140 IF ARRAY (J – 1) > ARRAY (J) THEN SWAP
ARRAY (J – 1), ARRAY (J): F = 1
```

Los programas de clasificación se pueden utilizar para todo tipo de cosas. Aunque la clasificación por el método de la burbuja no sea super-eficiente, es el sistema más adecuado para empezar a estudiarlas.

Funciones matemáticas

El MSX-BASIC proporciona un número de funciones específicas para operaciones matemáticas. No merece la pena que las estudiemos todas pero echaremos un vistazo a las más frecuentemente utilizadas. A una función se le suministra un valor y retorna un valor. Veremos las funciones trigonométricas **SIN**, **COS** y **TAN** y la que calcula la raíz cuadrada, **SQR**.

A una función hay que suministrarle lo que se conoce como **argumento**. Un argumento es el valor que Vd. da a la función para que lo

procese. El argumento va siempre entre paréntesis y después del nombre de la función.

La función SIN calcula el valor del seno de un ángulo. SIN toma un ángulo en radianes y retorna el seno de ese ángulo. Si Vd. no está familiarizado con los radianes, y prefiere utilizar grados, entonces utilice la siguiente ecuación para convertir un número de grados a radianes:

Radianes = grados*PI/180

PI es aproximadamente igual a 3,141593. Así pues para calcular el seno de 45 grados, convertimos el valor en radianes y aplicamos la función SIN:

```
10 A=45
20 R=A*(3.141593#/180)
30 PRINT SIN(R)
```

El resultado de la ejecución de este programa será la aparición del número 0.7071 en la pantalla. Las funciones COS y TAN trabajan también con radianes y retornan el valor del coseno y la tangente de un ángulo respectivamente.

La función SQR retorna la raíz cuadrada de un número siempre y cuando el valor que Vd. le dé no sea un número negativo o cero. Los ordenadores pueden hacer muchas cosas pero encontrar raíces cuadradas imposibles no es una de ellas. Pruebe el siguiente programa:

```
10 PRINT SQR(100)
20 PRINT SQR(4)
30 PRINT SQR(16)
```

Se visualizarán los valores 10, 2 y 4. En el Apéndice A se relacionan todas las funciones, matemáticas y de otro tipo, disponibles en el MSX-BASIC.

Funciones definidas por el Usuario

Si no encuentra en el MSX-BASIC la función matemática que Vd. precisa, tiene la posibilidad de crearla Vd. mismo. La sentencia DEF FN le permite agrupar una serie de sentencias y darles un nombre para referencias posteriores. Por ejemplo, Vd. quiere obtener el cuadrado de un

número mediante una función (si no existiese la posibilidad de utilizar X^2). La declaración de esta función sería:

10 DEF FNA(X) = X*X

La instrucción DEF FN tiene la siguiente sintaxis:

DEF FN <nombre de función> (<variable>, <variable>,...) = <expresión>

En nuestro ejemplo, el nombre de la función es A y la expresión es X*X. El valor escrito entre paréntesis se conoce como **variable falsa (dummy)** para la función. Las variables falsas quedan sustituidas por los valores actuales cuando Vd. utiliza (o **llama** a) la función. Intente ejecutar este sencillo programa:

```
10 DEF FNA(X)=X*X
20 INPUT S
30 PRINT FNA(S)
```

El programa obtendrá el cuadrado del número que Vd. teclee. En este caso X es la variable falsa de la función y es sustituida por el valor asignado a S en la línea 30. Vd. no está limitado a un sólo argumento para la función. Si quiere multiplicar dos valores arbitrarios, puede conseguirlo utilizando el siguiente programa:

```
10 DEF FNA(X,Y)=X*Y
20 INPUT S,T
30 PRINT FNA(S,T)
```

Las funciones definidas por el usuario se utilizan de la misma forma que las funciones soportadas por el MSX-BASIC.

¿Recuerda Vd. el Teorema de Pitágoras? Aquí está para aquellos que no lo recuerden. En un triángulo rectángulo, el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los otros dos lados. Así pues, si queremos calcular la longitud de la hipotenusa (c) de un triángulo rectángulo de lados (a) y (b) utilizaremos la siguiente ecuación:

$$c = \sqrt{a^2 + b^2}$$

El programa siguiente contiene una función adecuada para calcular dicha longitud:

```
10 DEF FNA(A,B)=SQR((A^2)+(B^2))
20 INPUT "LONGITUD DEL LADO A ";X
30 INPUT "LONGITUD DEL LADO B ";Y
40 PRINT "LA LONGITUD DEL LADO C ES ";
FNA(X,Y)
50 END
```

Utilizaremos una variante de esta función más adelante, en un programa de gráficos. Son muchas las ventajas que proporciona la utilización de funciones. Si está Vd. haciendo juegos malabares con las ecuaciones para dividir un átomo, el tecleo de unas ecuaciones tan largas va a resultarle muy pesado.

Una vez definidas, la utilización de las funciones es algo muy sencillo y conciso. También son más rápidas que la utilización interactiva de la misma expresión matemática. El intérprete BASIC no tiene que reprocesar lo que, en esencia, es la misma ecuación con diferentes variables. Con las funciones también se ahorra espacio de memoria.

Por último, facilitan la comprensión del programa, ya sea cuando Vd. lo vea, unas pocas semanas después de haberle escrito, ya sea cuando algún inocente ayudante se quede maravillado con su último programa y le pregunte (siempre fortuitamente), "Bueno, pero ¿cómo lo hace?..."

Cómo interactuar con sus programas

En esta sección, trabajaremos con unos cuantos programas muy útiles y ampliaremos los conocimientos sobre la validación de los datos y la forma de que las salidas producidas por sus programas sean realmente bonitas.

Entrada de Datos

Veamos primero algunos métodos para producir rutinas de entrada fáciles de utilizar, y para ello presentaremos el mandato LOCATE y la variable INKEYS en unos ejemplos prácticos de creación de programas dirigidos mediante un menú.

Un sistema dirigido por menú, cuando se aplica a los programas de ordenador es exactamente lo mismo que si Vd. se encuentra en un restaurante en el que Vd. utiliza el menú para seleccionar aquello que desea. Antes de que se pregunte qué es lo que tiene que ver la comida con los ordenadores, nos explicaremos. En un restaurante, Vd. selecciona del menú aquello que quiere comer y el camarero se va luego y se lo trae. Con el ordenador Vd. puede visualizar las operaciones en pantalla, seleccionar lo que desea y dejar que el ordenador realice la tarea que Vd. eligió. La forma más fácil de hacer esto es visualizar en pantalla un número junto con una opción y permitir que el usuario elija el número de la opción deseada. Este sencillo programa visualiza una lista de elementos en pantalla y pide al usuario que seleccione un número de opción.

```
10 REM MENU SENCILLO DEL SISTEMA
20 REM VISUALIZAR LISTA DE OPCIONES
30 CLS:PRINT "1. VISUALIZAR UN MENSAJE"
40 PRINT "2. EJECUTAR ALGUNAS NOTAS"
50 PRINT "3. DIBUJAR UN CIRCULO"
60 PRINT "4. SALIDA"
70 LOCATE 1,20
80 PRINT "SELECCIONE NUMERO DE OPCION"
```

```
90 A$=INKEY$:IF A$="" THEN 90 ELSE PRINT
 A$
100 OPT%=ASC(S$)-48
110 IF (OPT%<1) OR (OPT%>4) THEN BEEP:
LOCATE 22,20:GOTO 90
120 REM SELECCION DE LA ACCION A TOMAR
130 ON OPT% GOSUB 500,600,700,800
500 CLS:PRINT "HOLA. ¡SOY UN ORDENADOR
MSX!"
510 FOR I = 1 TO 800:NEXT I
520 RETURN 30
600 REM EJECUTAR ALGUNAS NOTAS
610 PLAY "CDEFG"
620 FOR I=1 TO 800:NEXT I
630 RETURN 30
700 REM DIBUJAR UN CIRCULO
710 SCREEN 2
720 CIRCLE (128,92),70,1
730 PAINT (128,92),1
740 FOR I=1 TO 800
750 SCREEN 1:RETURN 30
800 CLS:END:REM TERMINACION DEL PROGRAMA
```

En la línea 90 aparece la variable especial INKEY$. Esta variable almacena el nombre de una tecla, cuando se ha pulsado alguna. La sentencia de la línea 90 asigna el contenido actual de INKEY$ a la variable A$. Si A$ está vacía (IF A$ = " "), entonces es que no se ha pulsado ninguna tecla, el programa salta al comienzo de la misma línea y se mantiene analizando la variable INKEY$ hasta que contenga algún valor. Se conoce como **polling** a la investigación repetitiva para ver si se ha producido un suceso. La línea 90 está haciendo un **polling** al teclado.

Cuando se pulsa una tecla, se almacena su valor en A$. Lo que hay que hacer a continuación es preguntar si la tecla pulsada era "1", "2", "3" ó "4". Observe que estamos tratando todo lo que nos viene del teclado como si fuese un carácter. Podríamos haber utilizado una sentencia de entrada normal, pero esto dejaría al programa completamente abierto.

Vd. podría teclear una G, por ejemplo, con lo cual el intérprete BASIC produciría un mensaje de error. Los programas deben verificar por sí mismos los datos de entrada –el BASIC no necesitaría hacerlo.

Lo que el programa hace es convertir el carácter en un entero y entonces comprobar si es un dato válido. Una función BASIC, ASC, convierte los caracteres a enteros. Todos los caracteres del MSX-BASIC tienen un número de código, conocido como su código ASCII (Ver Capítulo 1). La función ASC obtiene este código de un carácter o de una cadena de caracteres. Los códigos ASCII de los números 1, 2, 3 y 4 son 49, 50, 51 y 52 respectivamente. Así pues, la línea 100 convierte la letra que se ha tecleado en un código ASCII y resta 48 al número de código. Si se pulsó la tecla 1, la línea 100 la convierte para obtener el número de código 49 y al restarle 48 queda justamente lo que queríamos —el número 1.

El programa entonces puede comprobar si el valor introducido es válido, por ejemplo, no debe ser menor que 1 ni mayor que 4. Si el valor fuese incorrecto se oye un pitido y podemos ver otra nueva característica del BASIC. El cursor se mueve desde donde se encuentre a una posición nueva dada por el mandato LOCATE. En este caso, el cursor se coloca al final de la línea "SELECCIONE NUMERO DE OPCION" listo para aceptar los datos corregidos. LOCATE 22, 20 significa "colocar el cursor en la posición del carácter 22 de la línea 20". En la modalidad de texto 0 la posición más alta para el carácter es 40, ya que ésta es la anchura máxima de la pantalla, en modalidad de texto 1, es 36. Vd. también puede posicionar el cursor en la modalidad de gráficos en alta resolución; la principal diferencia estriba en que, en este caso, se considera que la pantalla tiene 256 caracteres de ancho y un número máximo de 192 líneas.

En algún momento se introducirán los datos ya corregidos y el programa tendrá que decidir sobre lo que el usuario desea. Aquí es donde viene la sentencia ON...GOSUB. Es como tener una cadena completa de sentencias IF...THEN, todas en una línea. El BASIC mira la condición especificada, en este caso la elección del usuario, saltando a otra sección del programa dependiendo de si la condición es 1, 2, 3 ó 4. En lugar de utilizar la línea 130, se podría haber escrito la siguiente codificación:

```
130 IF OPT%=1 THEN GOSUB 500
132 IF OPT%=2 THEN GOSUB 600
134 IF OPT%=3 THEN GOSUB 700
138 IF OPT%=4 THEN GOSUB 800
```

Una vez que el ordenador ha hecho lo que el usuario quiere, se queda a la espera de que sea pulsada una tecla, antes de volver al menú principal. Esto se consigue haciendo un polling al teclado utilizando INKEY$. Cuando INKEY$ tiene un valor, el programa empieza de nuevo y el

proceso puede continuar hasta que el usuario seleccione la posición 4, con lo cual finaliza el programa.

Cómo utilizar un mando de juego

Una alternativa para utilizar un programa dirigido por menú es el empleo de un mando de juego para mover el cursor por la pantalla para apuntar a una opción. Una vez apuntada una determinada opción, se pulsa la barra espaciadora, quedando seleccionada dicha opción. Se puede escribir un programa sencillo que simplemente visualice una lista de palabras en la pantalla, permita al usuario mover el cursor por la pantalla, colocarle cerca de la opción deseada y que haga algo cuando se haya pulsado la barra espaciadora. En este ejemplo hemos introducido dos funciones nuevas, una variable especial y una nueva sentencia, todas ellas de gran utilidad. Para ver cómo funcionan, escriba este nuevo programa dirigido por menú:

```
10 REM MENU 2 DEL SISTEMA
20 REM ESTABLECER MODALIDAD DE PANTALLA
30 SCREEN 1:KEY OFF
40 REM INICIALIZAR X E Y
50 X=1:Y=1
60 REM VISUALIZAR EL MENU
70 LOCATE 18,12:PRINT "SALIDA"
80 LOCATE 18,16
90 PRINT "TONADA"
100 REM ACTIVAR DETECCION DE BARRA ESPAC
IADORA
110 STRIG(0) ON
120 REM COLOCAR EL CURSOR EN LA POSICION
 ESPECIFICADA POR X E Y
130 LOCATE X,Y,1
135 REM VISUALIZAR "X"
140 ON STRIG GOSUB 300:REM VER SI SE HA
PULSADO LA BARRA ESPACIADORA
150 REM SALTAR A LA DIRECCION APUNTADA P
OR LOS MANDOS DE JUEGOS (IF STICK(0)=3 T
HEN 200,ETC.)
160 A=STICK(0)
170 ON A GOTO 190,195,200,205,210,215,22
0,225
```

```
180 GOTO 160
190 Y=Y-1        :GOTO 240:REM ARRIBA
195 X=X+1:Y=Y-1:GOTO 240:REM ARRIBA Y A
LA DERECHA
200 X=X+1        :GOTO 240:REM DERECHA
205 X=X+1:Y=Y+1:GOTO 240:REM ABAJO Y A L
A DERECHA
210 Y=Y+1        :GOTO 240:REM ABAJO
215 X=X-1:Y=Y+1:GOTO 240:REM ABAJO Y A L
A IZQUIERDA
220 X=X-1:       :GOTO 240:REM IZQUIERDA
225 X=X-1:Y=Y-1:GOTO 240:REM ARRIBA Y A
LA IZQUIERDA
240 REM COMPROBAR QUE LAS COORDENADAS SO
N VALIDAS. SI NO LO SON RESTAURARLAS CON
 UN VALOR ACEPTABLE
250 IF X>40 THEN X=40
260 IF X<1 THEN X=1
270 IF Y>24 THEN Y=24
280 IF Y<1 THEN Y=24
290 GOTO 130:REM COLOCAR EL CURSOR EN LA
NUEVA POSICION
300 REM SUBRUTINA - VALIDA LAS OPCIONES
Y LAS EJECUTA
310 IF CSRLIN=12 THEN GOTO 500
320 IF CSRLIN=16 THEN PLAY "CDEFGGFEDC":
RETURN
330 REM SEÑAL DE ERROR CON UN BEEP
340 BEEP:RETURN
500 CLS:PRINT "FIN":END
```

La sentencia STRIG(0) ON dice al ordenador que compruebe si se ha pulsado el botón disparador del mando de juego (0). Si no existiese ningún mando de juego conectado al ordenador, se considera a la barra espaciadora como el botón disparador. La sentencia ON STRIG GOSUB le dice al ordenador qué es lo que debe hacer cuando se ha pulsado la barra espaciadora o el botón disparador. Cuando la subrutina ha hecho lo que Vd. quería que hiciese al pulsar el botón, se 'auto-ejecuta' otro STRIG(0) ON al retornar de la subrutina. Vd. también puede utilizar las sentencias STRIG(0) OFF y STRIG(0) STOP. La primera le dice al ordenador que no es necesario comprobar si se ha pulsado el botón, la

última sentencia le dice al ordenador que tome nota de que se ha pulsado un botón pero que no haga nada. El ordenador saltará inmediatamente a la subrutina que sabe qué es lo que hay que hacer, cuando se pulsan los disparadores, al ejecutarse una sentencia STRIG(0) ON y se haya pulsado anteriormente un disparador.

La función STICK comprueba la dirección en la que se mueve el mando de juego. Si el valor retornado por STICK(0) es 0 significa que el mando no se ha movido nada. Si Vd. no tiene conectado al ordenador ningún mando de juego, se pueden utilizar las teclas del cursor. Para obtener la dirección de la diagonal hacia la derecha y hacia arriba por ejemplo, se pulsarán las teclas Up (arriba) y Right (Derecha) simultáneamente. STICK (0) toma, por defecto, las teclas del cursor, a menos que Vd. conecte un mando de juegos al ordenador.

POS es una función que retorna un número indicando la posición horizontal (columna) actual del cursor. Vd. puede utilizar esta función con cualquier argumento. Como el argumento no influye para nada en la información que le devuelve la función, se le conoce como **argumento falso**. CSRLIN es una variable especial que contiene la posición vertical (fila) actual del cursor.

Cómo utilizar las interrupciones

Las interrupciones son un tipo de instrucciones muy útiles; son pocas y se encuentran en la mayoría de los dialectos del BASIC. Una interrupción como su nombre indica, es una ruptura del flujo normal de un programa causada por algún suceso. Una interrupción se utiliza para **detectar** un suceso —se percata de que ha tenido lugar algún suceso y a continuación hace algo relacionado con él.

Este sistema es diferente del sistema de **polling** que utilizamos antes para ver si se ha pulsado una tecla. Para ilustrar las diferencias, he aquí dos programas, diseñados ambos para ejecutar una nota cuando se pulse la barra espaciadora.

```
10 REM SISTEMA DE POLLING
20 PRINT "EL POLLING"
30 GOSUB 90
40 PRINT "ES MENOS"
50 GOSUB 90
60 PRINT "EFICIENTE"
70 GOSUB 90
80 GOTO 20
```

```
90 REM VER SI SE HA PULSADO LA BARRA ESP
ACIADORA
100 A$=INKEY$
110 IF A$=CHR$(32) THEN PLAY "C":RETURN
120 RETURN
```

Ahora el mismo problema resuelto con interrupciones:

```
10 REM SISTEMA DE INTERRUPCIONES
20 STRIG(0) ON
30 ON STRIG GOSUB 80
40 PRINT "LAS INTERRUPCIONES"
50 PRINT "SON MAS"
60 PRINT " EFICIENTES"
70 GOTO 40
80 PLAY "CDEFG":RETURN
```

Con el método del polling, se ha escrito una sección del programa para comprobar si se ha pulsado la barra espaciadora. En el segundo programa el mandato STRIG(0) dice al ordenador que compruebe si se está pulsando la barra espaciadora y la sentencia ON STRIG GOSUB determina a qué número de línea debe bifurcar el programa cuando se haya pulsado la barra espaciadora.

Para explicar las diferencias que existen entre los sistemas de polling y de interrupciones imagínese el siguiente escenario. Suponga que está Vd. preparando un pastel al horno, y al mismo tiempo, está escribiendo una carta a un amigo. Si Vd. se comportase como un sistema de polling, tendría que ir a mirar el horno cada cinco o diez minutos aproximadamente para ver si el pastel está ya listo para luego volver a la carta. Un sistema de interrupciones precisaría que el horno estuviera equipado con un reloj y una alarma. Vd. puede seguir escribiendo su carta sin tenerse que preocupar sobre cómo va el pastel hasta que suene la alarma del horno avisándole de que ya puede sacar el pastel.

En el MSX-BASIC hay disponible un amplio rango de interrupciones. Detectan sucesos como el tecleo de una tecla de función, la pulsación de la tecla STOP, errores, colisiones de duendes y similares. Para posibilitar las interrupciones existen dos mandatos —<interrupción> ON (que acabamos de ver) e <interrupción> STOP. <interrupción> STOP memoriza la ocurrencia de un suceso pero desactiva la sentencia ON <interrupción> GOSUB posterior. Con esto puede desactivar la detección de los sucesos por un tiempo, sin hacer ningún tratamiento del suceso cuando ha ocurrido. La ejecución del mandato <intervalo> ON

provocará la bifurcación del programa si se ha memorizado una interrupción.

Unas palabras para alertar sobre el uso de estas interrupciones. Cuando se ha ejecutado una sentencia ON ERROR, *todas* las otras interrupciones quedan desactivadas. Así que si Vd. utiliza la sentencia ON ERROR con otras interrupciones, tenga en mente que debe activar las otras interrupciones de nuevo en la rutina de tratamiento de la interrupción.

La utilización de estas instrucciones de interrupciones permiten la escritura de un código más conciso. Su principal desventaja es que los programas se hacen algo más difíciles de seguir y depurar por lo que deben estar muy bien comentados.

Salida de Datos

En los anteriores capítulos hemos visto el mandato PRINT y cómo se puede utilizar para producir salidas formateadas en pantalla, con la ayuda de puntos y puntos y comas. Ya comentamos, en su momento que veríamos con más detalle el mandato PRINT USING.

PRINT USING es, no obstante, una especie de batiburrillo del PRINT. Con esta sentencia se puede especificar el formateo de una salida de una manera determinada. Para hacer esto, hay que incorporar un juego de caracteres de formateo especiales. Para la salida de caracteres de datos existen tres caracteres de formateo, !, & y @. Vayamos primero con el signo de exclamación. Veamos qué pasa cuando se ejecuta el siguiente programa:

```
10 INPUT A$
20 PRINT USING "!";A$
```

Sólo se visualiza el primer carácter de la cadena. Esto puede ser particularmente útil, por ejemplo, para imprimir una lista de nombres. Este programa se puede utilizar para cortar los nombres por las iniciales. La entrada del programa son los nombres y se obtiene como salida la inicial del nombre y el apellido:

```
10 INPUT "NOMBRE ";C$
20 INPUT "APELLIDO ";S$
30 PRINT USING "!";C$;
40 PRINT ".";S$
50 GOTO 10
```

Si hubiésemos introducido los siguientes datos: "CARLOS SANZ", "ABEL DIMAS" y "LEON ABAD" la salida producida por el programa sería:

C. SANZ
A. DIMAS
L. ABAD

El segundo carácter de formateo para las cadenas es el signo &. Este carácter formateador se especifica de una manera algo distinta, ya que con:

```
10 PRINT USING "&&";"FRED";"ERIC"
20 PRINT USING "&   &";"FRED";"ERIC"
```

La salida aparecería así:

FRER
FRED ERIC

La regla que sigue este formateador es que por lo menos se visualizan dos caracteres de una cadena, más tantos caracteres como espacios haya entre los dos &. Así en el primer ejemplo sólo se obtienen los dos primeros caracteres, tanto de "FRED" como de "ERIC" dando como resultado "FRER". En el segundo ejemplo, hay tres espacios por lo que se visualizan los otros dos caracteres de "ERIC" y "FRED" además de un espacio más. Si hubiésemos especificado el siguiente formato:

20 PRINT USING "& &"; "FRED"; "ERIC"

Vemos que se produce la cadena "FREERI". El carácter formateador final es el símbolo @. Esto especifica que la cadena de caracteres debe visualizarse tal cual es. Por ejemplo, el siguiente programa toma su nombre de entrada y lo incluye en el mensaje que va a visualizar:

```
10 INPUT "CUAL ES SU NOMBRE ";N$
20 PRINT USING "HOLA @, ENCANTADO";
N$
```

Resultado: si Vd. teclea "JUAN PEREZ" el ordenador le contestará con:

"HOLA JUAN PEREZ, ENCANTADO"

También hay formateadores especiales para números y son #, +, –, **, ¥ ¥, **¥ y ^^^^. Algunos se utilizan muy poco, especialmente el que visualiza los signos del Yen. No obstante daremos una breve pasada para ver que es lo que hacen los más corrientes.

El símbolo # denota un número. Si Vd. quiere visualizar una serie de números de una forma nítida deberá utilizar este carácter formateador. Como ejemplo, escriba lo siguiente:

```
10 PRINT USING "### . ##"; 1.646, 134.5, .45, 6.91
```

El programa produciría la siguiente salida:

```
1.65    134.50    0.45    6.91
```

Si es preciso, los números quedan redondeados con este carácter formateador. Si se especifica un punto decimal y el número es fraccionario, se pondrá un cero siempre antes del punto decimal. Observe que también se insertan espacios delante del número si éste es más corto que el dado por la sentencia PRINT USING.

Los signos más y menos harán que aparezca un signo + o – delante o detrás del número:

```
10 PRINT USING "+###.##";12.86,-12.86
20 PRINT USING "###.##+";12.86,-12.86
30 PRINT USING "###.##-";12.86,-12.86
```

La salida obtenida será algo así:

```
+12.86    -12.86
 12.86     12.86-
 12.96     12.86-
```

El formateador "**" rellena los espacios de cabecera de un número con asteriscos:

```
10 PRINT USING "**#.##";12.86
20 PRINT USING "**#.##";1.83
30 PRINT USING "**#.##";123.67
```

La salida de este programa sería así:

```
*12.86
**1.83
123.67
```

Existen también otros caracteres de formateo que insertan el símbolo del Yen japonés delante de un número. A menos que tenga Vd. que trabajar con la divisa japonesa frecuentemente, es muy posible que apenas utilice para nada estos formateadores.

Hay versiones de estas sentencias PRINT y PRINT USING para trabajar con una impresora. Se llaman LPRINT USING y LPRINT y funcionan exactamente igual con la diferencia de que la salida va dirigida a la impresora (si el sistema tiene una conectada), en lugar de aparecer en la pantalla.

Entrada y Salida con Ficheros

Básicamente, un fichero es una colección de datos. MSX-BASIC proporciona una serie de instrucciones y variables especiales para su utilización con ficheros. Se pueden utilizar cuatro tipos de dispositivos. Sólo uno de estos dispositivos se puede utilizar tanto para almacenar como para recuperar los ficheros: la grabadora de cintas. Los otros tres dispositivos son solamente de salida. En la tabla que damos a continuación, aparecen los dispositivos disponibles, los nombres que el MSX-BASIC les da (su descriptor) y la forma en que pueden ser utilizados:

Nombre del dispositivo	Descriptor del dispositivo	Entrada/Salida
Grabadora de cintas	CAS:	Entrada y Salida
Impresora de líneas	LPT:	Salida solamente
Monitor/TV	CRT:	Salida solamente
Pantalla de gráficos	GRP:	Salida solamente

Pondremos un interés especial en la grabadora de cintas, ya que es fundamental para la salvaguarda de los programas y probablemente ya tenga Vd. alguna por su casa.

Para trabajar con un fichero, en MSX-BASIC, primero tiene Vd. que **abrir** el fichero. El siguiente programa simplemente solicita del usuario tres palabras y las almacena en cinta:

```
10 OPEN "CAS:FRED" FOR OUTPUT AS #1
20 INPUT A$
30 INPUT B$
40 INPUT C$
50 PRINT #1,A$
60 PRINT #1,B$
70 PRINT #1,C$
80 CLOSE #1
```

La línea 10 del programa abre un fichero en cinta llamado "FRED" y declara que va a ser utilizado por el programa como un fichero de salida. El número que lleva el signo # delante es el **número de fichero.** El número de fichero # 1 es utilizado para referenciar al fichero "FRED" a lo largo de todo el programa. Después de ejecutada la línea 10, se le indicará que pulse los botones de play y record de la grabadora de cintas.

La salida al fichero se efectúa mediante la sentencia de la línea 30. La sentencia PRINT # simplemente escribe en la cinta las cadenas de caracteres que Vd. haya introducido desde el teclado. La sentencia es, de hecho, exactamente igual que una sentencia PRINT normal. Es más, también hay disponible, en el MSX-BASIC, una sentencia PRINT # USING. La última sentencia del programa borra el fichero. Si no especifica ningún número de fichero, se cierran todos. La sentencia END produce el mismo efecto. Lo que el programa hace cuando cierra un fichero, es escribir un carácter en la cinta para señalizar el final del fichero. Este carácter corresponde a las teclas CTRL y Z pulsadas simultáneamente.

Ahora que ya tiene sus cadenas de caracteres salvadas en cinta, querrá recuperarlas en una fecha posterior. El siguiente programa tomará unas cadenas de caracteres, como entrada, desde el fichero en cinta "FRED".

```
10 OPEN "CAS:FRED" FOR INPUT AS #1
20 IF EOF(1) THEN GOTO 100
30 INPUT #1,A$
40 PRINT A$
50 GOTO 20
60 CLOSE
```

Se ha modificado la sentencia OPEN para que los datos, en esta ocasión, sean de entrada. **EOF** es una variable especial del MSX-BASIC. Cuando se está leyendo un fichero desde cinta, se está controlando la presencia del carácter CTRL-Z. Cuando este carácter CTRL-Z es encontrado, la variable EOF toma el valor –1. EOF es la variable de **Final de Fichero (End-Of-File).** Si el programa no efectúa la comprobación del cambio de valor del EOF a –1, entonces se produce un error "Input Past End".

Los casos sencillos que acabamos de ver utilizan dos de las tres formas posibles de utilización de los ficheros. Además de los modos INPUT (ENTRADA) y OUTPUT (SALIDA) tenemos también el modo APPEND (AÑADIR). Si se abre un fichero en modo APPEND, el fichero se lee, desde cinta, hasta que se encuentra el carácter CTRL-Z, y la

nueva información puede ser añadida, entonces, sobreescribiéndose el final del fichero.

El siguiente es un programa muy sencillo diseñado para crear un fichero en cinta para guardar nombres y números de teléfonos y para recuperar un número determinado.

```
10 CLS
20 PRINT "1. CREAR EL FICHERO"
30 PRINT "2. RECUPERAR LA INFORMACION"
40 PRINT:PRINT
50 INPUT "SELECCIONE OPCION (1 O 2)";A$
55 PRINT "ENTRE CONTADOR DE CINTA NUM.":
INPUT "NUM. : ";C
60 IF A$="1" THEN GOTO 100
70 IF A$="2" THEN GOTO 240
80 BEEP:GOTO 50
90 REM CREAR / ACTUALIZAR EL FICHERO
100 OPEN "CAS:TEL" FOR OUTPUT AS #1
110 CLS
120 PRINT "CREACION DEL FICHERO"
130 PRINT "--------------------"
140 FOR I=1 TO 800:NEXT I
150 CLS
160 PRINT "PARA SALIR TECLEE * EN LUGAR
DE NOMBRE"
170 PRINT
180 INPUT "ESCRIBA EL NOMBRE : ";N$
190 IF N$="*" THEN PRINT "FIN DE LA CREA
CION: PRINT "REBOBINAR CINTA A ";C:GOTO
340
200 INPUT "TECLEE NUM. : ";T$
210 PRINT:INPUT ""ESTA BIEN? (S/N) ";R$
220 IF R$="S" THEN PRINT #1,N$,T$
230 GOTO 150
240 REM RECUPERAR INFORMACION
250 OPEN "CAS:TEL" FOR INPUT AS #1
260 CLS:PRINT "EMPEZAR CINTA EN ";C
270 INPUT "TECLEE EL NOMBRE: ";N$
280 PRINT "BUSQUEDA DEL NUMERO DE ";N$
290 IF EOF(1) THEN 330
```

```
300 INPUT #1,A$,B$
305 PRINT A$,B$
310 IF N$=A$ THEN PRINT "EL NUMERO ES ";
B$:F=1
320 GOTO 290
330 IF F=0 THEN PRINT "NO EXIST NUM. PAR
A ";N$
340 CLOSE
350 END
```

El problema que surge con la utilización de sistemas de ficheros basados en cintas es que trabajan increíblemente L-E-N-T-O-S y son francamente inflexibles. MSX-DOS proporciona unas opciones de manejo de ficheros en BASIC mucho más rápidas y más útiles aprovechando todas las ventajas de la velocidad y la capacidad de los discos flexibles.

Para otras funciones de formateo tales como SPACES y TAB, vea el Apéndice A.

En el siguiente capítulo, estudiaremos otros aspectos más dinámicos de la programación, cómo explotar las capacidades musicales del ordenador MSX (o cómo llegar a ser muy impopular entre su familia y sus amistades *muy* rápidamente).

Música y sonido en el MSX

En un ordenador, se pueden generar sonidos de distintas formas. Hace algunos años, los programadores que trabajaban en los grandes ordenadores y en los miniordenadores solían generar música con sus impresoras de líneas de una forma muy complicada. Enviando determinadas secuencias a la impresora encontraron que podían producir ligeros cambios en los tonos que, benévolamente, podrían llamarse notas musicales. Con algo de imaginación se podían ejecutar tonadas, no siempre muy agradables, de esta forma tan poco ortodoxa.

Con la llegada posterior de los ordenadores caseros, se pueden originar notas musicales enviando señales eléctricas a un altavoz. Si los impulsos se envían con la suficiente rapidez, se produce un tono. En la mayoría de los casos, el BASIC controla esto utilizando la sentencia POKE. Esta coloca unos números en una posición de memoria que a su vez es utilizada por una rutina en lenguaje ensamblador para pulsar un altavoz muy rápidamente y producir tonos reconocibles.

Los micros MSX tienen unos medios más sofisticados para producir sus sonidos. Tienen un chip dedicado a este propósito y al cual se puede acceder a través de dos sentencias y un mandato: SOUND, PLAY y BEEP. Este chip es capaz de producir tres sonidos diferentes simultáneamente con una variedad de sonidos que van desde los silbidos, estruendos y alaridos tan familiares para los que practican con los juegos hasta las notas y acordes musicales.

Ya hemos visto antes el mandato BEEP. Recuerde cómo es su sonido, escribiendo BEEP. Horrible ¿no? Si Beethoven se hubiese tropezado con el mandato BEEP nos habríamos quedado lamentablemente, sin sinfonías. Las sentencias realmente útiles son SOUND y PLAY.

La sentencia SOUND es la más difícil de aprender de las dos, sin embargo, proporciona una mayor flexibilidad en cuanto a los tipos de sonido que puede Vd. producir. Lo que la sentencia SOUND hace es enviar unos números directamente al chip de sonido. Este chip tiene 13 registros, todos ellos accesibles. Cada registro almacena un número que, controla una función específica del chip. Estas funciones se muestran en la tabla siguiente. Dependiendo del registro al que se le envía un número, y el valor de ese número, suceden determinadas cosas. ¿Confuso?

Bueno, para ayudarle a clarificar las cosas, acerquémonos al **modelo de programas** del chip de sonido del MSX, el General Instruments' AY-3-8910 (sonora ronda de aplausos, por favor).

Núm. REGISTRO	FUNCION
0	Controla el ajuste fino de tono para el sonido del altavoz A. Se utilizan los 8 bits de forma que este registro admite cualquier número comprendido entre 0 y 255.
1	Controla el ajuste grueso del canal A. Sólo se utilizan 4 bits por lo que el máximo valor que puede albergar este registro es 15.
2	Ajuste de tono fino para el canal B.
3	Ajuste de tono grueso para el canal B.
4	Ajuste de tono fino para el canal C.
5	Ajuste de tono grueso para el canal C.
6	Modula el canal de ruido. Se utilizan 5 bits por lo que los valores están comprendidos entre 0 y 63.
7	Este registro puede cambiar la salida de un canal de sonido de un tono a un ruido semejante al silbido. Para valores inferiores o igual a 7 se produce un tono desde los canales. Para valores superiores a 7 se producirá ruido desde los tres canales.
8	Establece el volumen del canal A.
9	Establece el volumen del canal B.
10	Establece el volumen del canal C.
11	Tono fino 8 bits.
12	Tono grueso 8 bits.
13	Controlador de envolvente.

Registros de sonido

El modelo de programación de una pieza hardware le indica al programador qué es lo que está controlando cada uno de los registros del chip. Esto se demuestra con el siguiente programa:

```
10 REM DEMOSTRACION DEL SONIDO
20 REM INICIALIZAR SONIDO CANAL A. ESTAB
LECER TONO GRAVE CON 1 Y TONO FINO CON 0
30 SOUND 0,1
40 REM INICIAR EL SONIDO PARA EL CANAL A
```

```
50 SOUND 8,5
60 END
```

Si ejecuta este programa observará que suceden dos cosas. Una es un espantoso quejido proveniente del altavoz de su televisor, y la otra es que aparece el mensaje de diálogo "OK" en pantalla indicando que el programa ha finalizado su ejecución. "¡Pero bueno!" oigo su pregunta "¿por qué no se para este espantoso quejido cuando se para el programa?". La razón por la cual sigue sonando persistentemente es porque el programa le ha dicho al chip de sonido que produjese ese ruido, pero no le ha dicho que pare de hacerlo. Puede Vd. librarse de este ruido tan horrible de dos formas. Una es escribiendo BEEP –ya que BEEP utiliza el chip de sonido, restaura todas las cosas y termina el sonido.

La otra forma de eliminar este potencial dolor de cabeza es tecleando SOUND 8,0. Con esto se baja el volumen del canal A a cero. Ahora, ¿cómo podía producir el programa un sonido tan horrible? Al colocar los valores 0 y 1 en los registros 0 y 1 del chip del sonido, el programa estaba estableciendo el tono del sonido para el canal A. Al asignar un 5 al registro 8, el programa estaba elevando el volumen del canal A. Pruebe con el siguiente programa y vea qué es lo que pasa:

```
10 SOUND 0,1:SOUND 8,5
20 FOR I=1 TO 14:FOR J=1 TO 100:NEXT J
30 SOUND 1,I
40 NEXT I
50 SOUND 8,0
60 END
```

Este programa produce 14 tonos diferentes. Habrá observado que el sonido ha finalizado con el programa.

Seamos más audaces y conectemos todos los canales de sonidos a la vez. Esto producirá un acorde:

```
10 SOUND 0,1:SOUND 1,1
20 SOUND 2,1:SOUND 3,3
30 SOUND 4,1:SOUND 5,6
40 FOR I=8 TO 10
50 SOUND I,5
60 NEXT I
```

Estas son las formas básicas de producción de notas musicales. Lo más excitante de la sentencia SOUND es que Vd. produce muchísimos sonidos extraños. La forma más sencilla es utilizando los registros de tono fino.

Ejemplos de Programas con SOUND

La mejor manera de comprender el chip de sonido es practicando, por eso le hemos dado una serie de programas diferentes para que pruebe con ellos. Le aconsejamos encarecidamente que haga sus propias modificaciones a los programas, para ver los efectos que producen.

Esta parte del libro transcurre entre ruidos extraños. Avise con anticipación a todo el mundo...

```
10 REM SONIDOS EXTRAÑOS
20 INPUT N
30 SOUND 0,1:SOUND 1,1:SOUND 8,5
40 FOR I=255 TO 0 STEP -N
50 SOUND 0,I
60 NEXT I
70 GOTO 40
```

Intente con valores de N como 5, 15, . 5 etc. y vea qué ocurre.

El canal de ruido tiene enormes posibilidades. El registro 7 se utiliza para activar el canal de ruido. He aquí dos programas, ambos pretenden producir un sonido parecido al de los helicópteros.

```
10 SOUND 7,5
20 SOUND 8,7
30 FOR I=63 TO 1 STEP-1
40 SOUND 6,I
50 NEXT I
60 GOTO 30
```

¡Y ahora otro ruido de helicóptero más el ruido del motor!

Valor de S	Envolvente producida
0,1,2,3,9	
4,5,6,7,15	
8	

Valor de S	Envolvente producida
10	
11	
12	
13	
14	

Envolventes sonoras disponibles del mandato "S"

```
10 SOUND 0,20:SOUND 1,0:SOUND 2,30
20 SOUND 3,0:SOUND 4,0:SOUND 5,9
30 SOUND 6,0:SOUND 7,48:SOUND 8,16
40 SOUND 9,4:SOUND 10,6:SOUND 11,100
50 SOUND 12,2:SOUND 13,12
60 GOTO 30
```

Este pequeño programa hace uso de la capacidad de envolventes del generador de sonido. Una envolvente determina la curva de la forma de onda del sonido generado. Se pueden producir 8 envolventes distintas. En el diagrama de la figura aparecen estas formas de ondas y el valor que hay que dar al registro 13 para producirlas.

SOUND es una de esas sentencias con las que tiene Vd. que jugar mucho para llegar a conocerla bien. Si Vd. está interesado en la música, todos los aspectos involucrados con la sentencia SOUND le llevarán mucho tiempo y le obligarán a pensar en términos de números en lugar de notas. ¿No sería mucho más sencillo decirle al ordenador que ejecute la nota C o E?

El Macro Lenguaje Musical

El mandato PLAY le permite hacer esto y algo más. Vd. puede especificar las notas a ejecutar, sus longitudes, formas de onda y volúmenes. Todas las instrucciones dirigidas al chip de sonido aparecen como cade-

nas de caracteres. Como un ejemplo sencillo, la siguiente sentencia producirá un acorde.

```
PLAY "C","A","E"
```

El canal A ejecutará la nota C, el canal B ejecutará la nota A y el canal C ejecutará la nota E. El tono de las notas producidas puede variar 8 octavas de C a C. Las cadenas de caracteres contienen instrucciones que son reconocidas por el mandato PLAY. Esto constituye un pequeño lenguaje dentro del BASIC y es conocido como el **Macro Lenguaje Musical**. A continuación sumarizamos las instrucciones del Macro Lenguaje Musical (o MML).

Mandato MML	Función
A a G	Ejecuta la nota indicada en la octava actual. Se permiten también las opciones #, + y –, donde # y + indican un sostenido y – indica un bemol. Estas opciones, sin embargo, sólo están permitidas cuando corresponden a una tecla negra del piano. B #, por lo tanto, no es una nota válida.
O<n>	Establece la octava actual, donde n puede variar de 1 a 8. Cada octava va desde C a B y la octava 4 será la elegida por el MML a no ser que Vd. utilice la instrucción O.
N<n>	Proporciona una alternativa para especificar notas y octavas. n puede estar comprendido entre 0 y 96, donde 0 significa silencio, 1 significa la nota C de la octava más baja y así sucesivamente.
L<n>	Establece la duración de las notas siguientes siendo la longitud de la nota, 1/n de manera que: **L1** nota completa (redonda) **L2** media nota (blanca) **L3** una de un tresillo de blancas (1/3 de un compás de 4 por 4) **L4** cuarto de nota (negra) **L5** una nota de quintillo (1/5 de un compás) **L6** una de un tresillo de negras **L64** semifusa La longitud también puede seguir a una nota si Vd. desea cambiar la longitud de una nota solamente. Por ejemplo L2A y A2 son equivalentes. El valor por omisión es 4.
R<n>	Establece la duración de un silencio (o descanso) donde n puede variar de 1 a 64 y funciona exactamente igual que el mandato L. El valor por omisión es 4. A. después de una nota hace que la nota se ejecute con puntillo, es decir, 3/2 veces su duración normal. Después de la nota puede aparecer más de un punto ajustándose la longitud a ellos. Por ejemplo,

A... sonará 27/8 veces su duración normal. Los puntos también pueden aparecer después de un silencio para ajustar su duración de la misma forma.

T<n> — Establece el tiempo para el tono especificando el número de cuartas notas que se van a ejecutar en un minuto, n varía entre 32 y 255. El defecto es 120.

V<n> — Establece el volumen con n variando entre 0 y 15. El valor por defecto es 8.

M<n> — Establece el periodo de la envolvente con n variando de 1 a 65535 y un valor por defecto de 255.

S<n> — Establece la curva de la envolvente. n puede variar entre 1 y 15 con un defecto de 1.

X<variable de tipo cadena> — ejecuta una cadena especificada de notas, pausas, etc.

En los mandatos MML, el valor de n puede ser una constante o una variable. Si Vd. utiliza variables, debe asignarlas en la cadena de mandatos. Pruebe, por ejemplo, el siguiente programita:

```
10 INPUT X
20 PLAY "o=X;C"
30 GOTO 10
```

Observe el punto y coma después del nombre de la variable. Si se olvida Vd. de ello, el BASIC le devolverá un mensaje de error "ILLEGAL FUNCTION CALL". El programa ejecutará la nota C en la octava que Vd. indique.

Aquí damos una corta melodía para que la siga con los dedos:

```
10 PLAY "L4O5AECO4","L4O3AEBA","L8AECO3A
O4AECO3A"
20 PLAY "L4O5BECO4G+","L4O2G+","L8O3G+O4
CEBBECO3G+"
30 PLAY "L4O6CO5ECO4G","L4O2G","L8O4GO5C
EO6CCO5ECO4G"
40 PLAY "L4O5F+DO4AD","L4O3D","L8O5F+DO4
ADO5F+DO4AD"
50 GOTO 10
```

Como puede ver, se ponen en juego las tres voces y se ha variado la longitud de las notas. El siguiente programa genera notas pseudo-aleatorias utilizando la función RND.

```
10 REM GENERADOR DE NOTAS PSEUDO-ALEATOR
IAS
20 PITCH=INT(RND(10)*100)
30 IF (PITCH>96) THEN GOTO 10 ELSE PLAY
"S8M1500L24N=PITCH;":GOTO 20
```

La función RND es como una rueda de la fortuna. Cuando se utiliza la función se obtiene un valor comprendido entre 0 y 1. Decimos que el programa es pseudo-aleatorio porque producirá la misma secuencia de números aleatorios cada vez que se ejecute el programa. En este caso, se utilizan las instrucciones S y M en la cadena PLAY. S especifica el tipo de la envolvente sonora producida y M especifica el tiempo que la envolvente cubrirá las notas que van a ejecutarse.

El siguiente programa utiliza las tres voces para producir acordes al azar. El mandato de juego (joystick) es utilizado para incrementar la velocidad de ejecución de las notas incrementando o decrementando la longitud de la nota.

```
10 REM GENERADOR DE NOTAS PSEUDO-ALEATOR
IAS CON TRES VOCES Y CONTROL DE VELOCIDA
D
20 REM LONGITUD DE NOTA POR DEFECTO
30 L=12
40 REM POLL AL MANDO DE JUEGOS
50 IF STICK(0)=1 THEN L=L+1:IF L>64 THE
N L=64
60 IF STICK(0)=5 THEN L=L-1:IF L<1 THEN
 L=1
70 REM CALCULAR NUMEROS AL AZAR Y DESPLA
ZAMIENTOS PARA VOCES DOS Y TRES
80 PIT1=(INT(RND(10)*100):IF (PIT1>86)
OR (PIT1<11) THEN GOTO 80 ELSE PIT2=PIT
1-10:PIT3=PIT1+10
90 REM EJECUTAR UN ACORDE AL AZAR
100 PLAY "S8M1500L=L;N=PIT1;","S8M1500L=
L;N=PIT3;"
110 GOTO 50
```

El último programa de esta sección utiliza la instrucción X. Observará que se han declarado tres variables de tipo cadena al comienzo del programa. En el momento oportuno, se ejecutan dichas cadenas utilizando el mandato X.

La instrucción X permite la ejecución de cadenas musicales dentro de una sentencia PLAY. De esta forma, si Vd. tiene secuencias de notas que se repiten en una pieza musical, no necesita escribir dicha secuencia cada vez que se produzca. Por ejemplo, supongamos que desea Vd. repetir la siguiente secuencia de notas: "04BGB05CEG"; Vd. puede asignar el valor de esta cadena a una variable, la variable A$, y llamar a esta cadena por este nombre, mediante la instrucción X desde sentencias PLAY posteriores. El siguiente programita muestra cómo se utiliza la instrucción X.

```
10 A$="O4BGØ5CEG"
20 PLAY "XA$;"
30 END
```

Observe que se requiere la presencia del punto y coma después del nombre de la variable de tipo cadena.

```
10 A$="L24o5bgbo6cego5bgbo6cego5bgbo6ceg
"
20 B$="L24o4bgbo5cego4bgbo5cego4bgbo5ceg
"
30 C$="L24o3bgbo5cego4bgbo5cego4bebo4ceg
"
40 FOR I=1 TO 4
50 PLAY "v4XA$;","v4XB$;","v4XC$;"
60 NEXT I
70 FOR I=1 TO 2
80 PLAY "v10L103d","v9L24XA$;","v10L1o2a
"
90 PLAY "o3D",A$,"o2g"
100 PLAY "o3C",A$,"o2e"
110 PLAY "o3c",A$,"o2a"
120 NEXT I
130 FOR I=1 TO 2
140 PLAY "v12L12o3daco2go3ga","v1012o4c"
,"v1012o5c"
150 PLAY "14o4dg","o3b","o5g"
160 PLAY "L24o4cdefgbo5cdefgb","o3g","o5
g"
170 PLAY "o4bgbo5cego4bgbo5ceg","o3f","o
5f"
180 PLAY "sm1500o3ccccccco5cccccc","o4c",
"o5c"
```

```
190 PLAY "l4o4dg","o3b","l2o5d"
200 PLAY "l12cdefgb","l8o6dco5af","o5g"
210 PLAY "o4bgbo5ceg","sm15000l24o7ccccg
gggdddd","f"
220 NEXT I
230 PLAY "L24o4cdefgbo5cdefgb"
240 PLAY "L24o3cdefgbo4cdefgb"
250 PLAY "L24o2cdefgbo3cdefgb"
260 PLAY "v10l12o3g","v10l12o4b","v10l12
o4d"
270 PLAY "o3a","o4e","o5c"
280 END
```

Con el siguiente programa intentamos la ejecución de la introducción al "Coro de la Alegría". Vd. podrá distinguir un eco en algunas secuencias de notas (líneas 120-190).

```
10 A$ = "R8L40T255DR24DR24T120L8ER24D
20 B$ = "T120R18L12DR24ER24D"
30 C$ = "R8L6O3GR18AR18BO4L6CR24O3GO4C"
40 PLAY "T120L6G.","T120L6G.","T120L6G."
50 PLAY B$,"R18XB$;","R16XB$;"
60 PLAY A$,"R18XA$;",A$
70 PLAY A$,"R18XA$;",A$
80 PLAY "R6","R6","R6"
90 PLAY "T120L6G.","T120L6G.","T120L6G."
100 PLAY B$,"R18XB$;","R16XB$;"
110 PLAY A$,"R18XA$;",A$
120 PLAY A$,"R18XA$;",A$
130 PLAY C$,"R18XC$;",C$
140 PLAY A$,"R18XA$;",A$
150 PLAY A$,"R18XA$;",A$
160 PLAY C$,"R18XC$;",C$
170 PLAY A$,"R18XA$;",A$
180 PLAY A$,"R18XA$;",A$
```

La siguiente pieza musical es una sencilla secuencia de acordes.

```
10 FOR I=1 TO 4
20 PLAY "L12O3D","O4F+","O4A'
30 PLAY "O3D","O4G","O4B"
```

```
40 PLAY "O3D","O4A","O4C"
50 PLAY "O3D","O4G","O4B"
60 NEXT I
70 FOR I=1 TO 2
80 PLAY "O3G","O4B","O5D"
90 PLAY "O3G","O5C","O5E"
100 PLAY "O3G","O5D","O5F"
110 PLAY "O3G","O5C","O5E"
120 NEXT I
130 FOR I=1 TO 2
140 PLAY "O3D","O4F+","O4A"
150 PLAY "O3D","O4G","O4B"
160 PLAY "O3D","O4A","O4C"
170 PLAY "O3D","O4G","O4B"
180 NEXT I
190 FOR I=1 TO 2
200 FOR J=1 TO 4
210 PLAY "L12O5EC+O4A","O6E","O5C+"
220 NEXT J
230 FOR J=1 TO 4
240 PLAY "O5DO4BG","O6D","O5B"
250 NEXT J
260 NEXT I
270 PLAY "O3D","O4F+","O4A"
280 PLAY "O3D","O4G","O4B"
290 PLAY "O3D","O4A","O4C"
300 PLAY "O3D","O4G","O4B"
```

El atractivo del MML radica en su similitud con la notación musical normal. Todos aquellos que tengan conocimientos musicales se familiarizarán con el lenguaje muy rápidamente y los que no los tengan encontrarán que son capaces de componer música con una gran facilidad.

Gráficos en MSX-BASIC

Una de las cualidades más atrayentes de un ordenador es su posibilidad de realizar dibujos. Uno de los métodos de dibujo más apreciados por todos los programadores es producir un fichero conteniendo texto de forma que parezca un dibujo. ¡Cuántos dibujos de Snoopy se han obtenido en instalaciones de ordenadores de todo el mundo! Utilizando la impresora para sobre-imprimir, se pueden producir efectos de diferentes sombreados. El ejemplo más famoso de esta forma de arte del ordenador puede ser la versión de la *Mona Lisa* de Leonardo Da Vinci. Si Vd. tiene una impresora, puede experimentar con esta forma de dibujar con el ordenador. ¡Aunque tiene el inconveniente de que puede destrozar su impresora! Existe un sistema mucho más sencillo para la obtención de dibujos.

Todos los ordenadores MSX tienen un chip dedicado a la producción de imágenes de gráficos en la pantalla. Se puede acceder a este chip de distintas formas. Un método consiste en enviar datos directamente al procesador del vídeo mediante VPOKE (ver capítulo 2). Generalmente esto consume tiempo de proceso y desde luego no es para cardíacos. Una forma mucho más amena de obtener dibujos en la pantalla es utilizando los comandos gráficos internos del MSX-BASIC.

Como Vd. ya sabe, hay dos modalidades gráficas: en baja y alta resolución. La que más utilizaremos en este capítulo es la pantalla en alta resolución: SCREEN 2. Esta pantalla se configura con un total de 49152 puntitos llamados pixels que están colocados formando una malla de 256 pixels de ancho por 192 de largo. Inicialmente, cuando se empieza a trabajar con la modalidad de pantalla en alta resolución, todos los pixels están puestos a un color (el color de fondo).

Gráficos utilizando PSET y PRESET

Los mandatos de gráficos más sencillos son PSET y PRESET. PSET enciende un pixel, PRESET apaga un pixel. Se puede especificar el color del pixel mediante la opción de los mandatos PSET y PRESET. El siguiente programita enciende todos los pixels de la pantalla.

```
10 SCREEN 2
20 FOR I=0 TO 256
30 FOR J=0 TO 192
40 PSET(I,J),15
50 NEXT J
60 NEXT I
```

El color de todos los pixels es blanco (color 15). Habrá observado que la posición de los pixels en la pantalla viene dada por dos coordenadas. Con PSET y PRESET se pueden seleccionar y encender determinados puntos mediante sus coordenadas individuales, y de esta forma se pueden dibujar las líneas y figuras que se deseen. Una forma muy sencilla de seleccionar los pixels que queremos encender es utilizando el mando de juego. El programa que damos a continuación demuestra cómo se pueden encender determinados pixels de la pantalla utilizando el mando de juego como si fuese un lápiz.

```
10 REM PROGRAMA DE DOODLE
20 SCREEN 2
30 REM ESTABLECER COORDENADAS INICIALES
40 X=128 : Y = 96
50 STRIG(0) ON
60 D = 1 : E = 0
70 ON STRIG GOSUB 200
80 IF D = 1 THEN PSET (X,Y),15 ELSE LOCA
TE X,Y
90 REM POLL A LOS MANDOS DE JUEGOS
100 IF STICK(0)=1 THEN Y=Y-1
105 REM 6-158
110 IF STICK(0)=2 THEN Y=Y-1:X=X+1
120 IF STICK(0)=3 THEN X=X+1
130 IF STICK(0)=4 THEN X=X+1:Y=Y+1
140 IF STICK(0)=5 THEN Y=Y+1
150 IF STICK(0)=6 THEN Y=Y+1:X=X-1
160 IF STICK(0)=7 THEN X=X-1
170 IF STICK(0)=8 THEN X=X-1:Y=Y-1
180 GOTO 80
200 REM INVERTIR VALOR DE D
210 SWAP D,E : RETURN
```

La variable D permite al usuario del programa moverse por la pantalla sin dibujar nada. Cuando se pulsa la barra espaciadora el valor de D se cambia de 0 a 1 ó de 1 a 0, mediante la sentencia SWAP. Si el valor actual de D es 1 entonces los pixels se iluminan, si el valor es 0, no se ilumina ningún pixel pero se modifica la posición del cursor.

Vistos los mandatos PSET y PRESET ya estamos en situación de construir nuestro primer programa "con movimiento". Veremos brevemente cómo se puede simular el movimiento en la pantalla utilizando estos mandatos.

El principio de toda animación consiste en mostrar un dibujo, cambiarlo de alguna forma y mostrar la nueva figura —todo ello tan rápidamente que el ojo humano cree que está viendo algo en movimiento. De esta forma podemos simular el movimiento, utilizando PSET y PRESET.

Primeramente se fija un punto de la pantalla (por ejemplo, PSET(10, 10). A continuación se ilumina un punto adyacente, de la misma forma, mientras que se apaga el punto anterior (utilizando PRESET). Lo que el programa hace es repetir el proceso de encender y apagar los correspondientes puntos mientras que esté simulando el movimiento. Un bucle FOR...NEXT proporciona la continuidad del movimiento.

```
10 SCREEN 2
20 FOR I=10 TO 250
30 PRESET (I-1.100)
40 PSET (I,100)
50 NEXT I
60 FOR I=250 TO 10 STEP -1
70 PRESET (I+1,100)
80 PSET (I,100)
90 NEXT I
100 GOTO 20
```

Este programa lanzará un punto a lo largo de la pantalla de izquierda a derecha y hacia atrás de nuevo. Observe que los movimientos hacia la izquierda y hacia la derecha se describen en bucles FOR...NEXT diferentes. Mire también la sintaxis del PRESET. En el movimiento de izquierda a derecha hay que apagar el punto que hay detrás del punto al que se le hace PSET —así pues el PRESET se hace al I-1, 100.

Como en todos los programas, haga experimentos. Por ejemplo, intente que el ordenador emita un "beep" cuando el punto cambie de dirección. Intente que el punto se mueva sobre una línea diferente cada vez que cambie de dirección (¡aquí tendrá que incluir una nueva variable!).

LINE es otro mandato de gráficos muy útil. Con LINE se especifican un par de coordenadas iniciales y un par de coordenadas finales, dibujándose una línea que une ambos puntos, por ejemplo:

```
10 SCREEN 2
20 LINE(50,50)-(100,50),15
30 LINE(100,50)-(100,100),15
40 LINE(100,100)-(50,100),15
50 LINE(50,100)-(50,50),15
60 GOTO 60
```

El programa anterior dibuja un rectángulo. Se puede dibujar un rectángulo de una forma muy sencilla, con LINE, ya que sólo requiere una sentencia LINE. Como ya ha visto antes, si Vd. especifica las coordenadas de la esquina superior izquierda del rectángulo como punto inicial y la esquina inferior derecha como el punto final, esperará que se dibuje una línea diagonal. Esto es lo que sucede normalmente, si no se utiliza la opción B en la sentencia LINE (ver Capítulo 2). Teclee y ejecute este programa:

```
10 SCREEN 2
20 LINE(20,20) (40,40),,B
30 GOTO 30
```

La B que aparece al final de la sentencia LINE indica que lo que Vd. quiere es un rectángulo en lugar de una línea recta. LINE también permite colorear el rectángulo una vez dibujado. Para ello, escriba BF al final de la sentencia LINE, en lugar de B. Esto indica al ordenador que dibuje el rectángulo y después lo "pinte".

El listado que damos a continuación corresponde a un programa que dibuja rectángulos de tamaños aleatorios y los pinta con colores aleatorios.

```
10 SCREEN 2
20 DEFINT A-Z
30 V=INT(RND(1)*256):W=INT(RND(1)*192)
40 X=INT(RND(1)*192):Y=INT(RND(1)*192)
50 C=INT(RND(1)*15)
60 LINE(V,W)-(X,Y),C,BF
70 GOTO 30
```

Hemos hablado bastante de rectángulos. También podemos dibujar círculos, como ya hemos visto. He aquí algunos programas que dibujan unas paternas interesantes utilizando la sentecia CIRCLE.

```
10 REM ROLLO DE PAPEL
20 REM INTRODUCIR VALORES
30 INPUT "ESPACIOS DE LAS X ";X
40 INPUT "ESPACIOS DE LAS Y ";Y
50 INPUT "DECREMENTO DEL RADIO ";Z
55 SCREEN 2
60 FOR I=1 TO 256 STEP X
70 FOR J=1 TO 192 STEP Y
80 FOR K=100 TO 5 STEP -Z
90 CIRCLE (I,J),K,15
100 NEXT K
110 NEXT J
120 NEXT I
130 GOTO 130
```

Para empezar, pruebe introduciendo 50, 30 y 10. Obtendrá unos bonitos efectos parecidos a un rollo de papel con el programa. Aquí hay otro garabato:

```
10 REM GARABATEO DE CIRCULOS
20 SCREEN 2
30 FOR I=1 TO 192
40 CIRCLE (128,I),I,15
50 NEXT I
60 GOTO 60
```

Desde luego Vd. no está limitado a dibujar círculos. Puede especificar una elipse utilizando la opción aspecto (relación) de CIRCLE (que ya vio antes). Vd. puede especificar la relación de la altura con la anchura del círculo. Si por ejemplo, Vd. elige el número 1/3, el resultado será una elipse aplastada, y si Vd. elige un valor de 3/2 la elipse será más alta que ancha. El siguiente programa presenta la sentencia CIRCLE con diferentes valores de la opción aspecto:

```
10 SCREEN 2
20 FOR I=1 TO 32
30 CIRCLE(128,96),70,15,,,I/8
40 NEXT I
50 GOTO 50
```

Por último, Vd. puede dibujar un arco con CIRCLE. Lo que tiene que dar a la sentencia CIRCLE es un ángulo inicial y un ángulo final, en radianes, para el arco. Trabaje con este programa:

```
10 SCREEN 2
20 CIRCLE(128,92),70,15,5,4
30 CIRCLE(128,92),50,15,4,3
40 CIRCLE(128,96),40,15,3,2
50 GOTO 50
```

Vd. verá tres círculos incompletos. Observará que están incompletos en diferentes partes y esto es debido a que los ángulos inicial y final son distintos para cada caso.

Una vez que Vd. ha dibujado un círculo, elipse o rectángulo, puede colorearlo utilizando la sentencia PAINT. La única regla a considerar en la modalidad en alta resolución, cuando se está pintando una figura, es que el color utilizado *debe* ser el mismo que el utilizado en primer lugar para dibujar la figura. Este programa dibuja un círculo y lo pinta en negro.

```
10 SCREEN 2
20 CIRCLE(128,96),70,1
30 PAINT(128,96),1
40 GOTO 40
50 GOTO 50
```

Intente pintar el círculo en blanco en lugar de en negro. La pantalla aparecerá barrida por el color blanco porque el ordenador espera detener la pintura cuando alcance una frontera blanca. En la modalidad en baja resolución, PAINT pinta cualquier figura, independientemente del color de su borde.

A propósito del color, Vd. puede conocer cuál es el color de un pixel determinado de la pantalla, mediante la función POINT. Esta función retorna el número de color de un pixel dado por sus coordenadas. POINT podría utilizarla para ver si un misil ha caído sobre un objetivo. Todos los objetivos se pueden pintar con un color, negro por ejemplo. Cuando las coordenadas del misil coincidan con las de un pixel negro, entonces el misil ha acertado. Esta es una demostración sencilla de la sentencia POINT:

```
10 SCREEN 2
20 CIRCLE(50,50),25,15
30 CIRCLE(100,100),25,8
40 CIRCLE(150,150),25,1
50 PAINT(50,50),15:REM BLANCO
60 PAINT(100,100),8:REM ROJO
70 PAINT(150,150),1:REM NEGRO
```

```
80 FOR I=1 TO 5000:NEXT I:REM ESPERAR UN
  POCO
90 X=POINT(50,50)
100 Y=POINT(100,100)
110 Z=POINT(150,150)
120 SCREEN 1
130 PRINT X,Y,Z
```

En la pantalla se visualizarán los números 15, 8 y 1.

Duendes (Sprites)

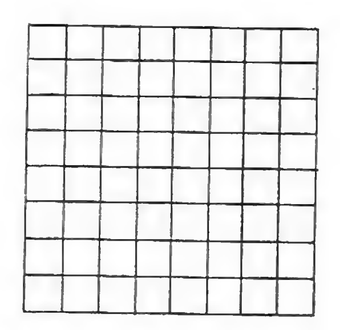
Los duendes (sprites) son unas traviesas figuritas que Vd. puede definir y mover en las dos pantallas de gráficos. Dan la impresión de existir totalmente independientes del resto de figuras de la pantalla dibujadas por sentencias como CIRCLE, LINE, PSET, etc. Los duendes, por razones que explicaremos enseguida, se han reservado para aplicaciones de programas de juegos, en los que pueden representar misiles, bombas, naves espaciales y todas las curiosas figuras que parecen estar al acecho en muchos juegos.

Primero, algo relacionado con la terminología empleada con los duendes que vamos a utilizar. Vd. puede considerar las pantallas de gráficos como un tablero en el que Vd. puede colocar sus dibujos utilizando los mandatos corrientes de gráficos. Se puede considerar a la pantalla de gráficos como un plano o superficie sobre la cual Vd. puede realizar sus dibujos. A este plano se le conoce como **plano cero**. Ahora imagine que el tablero de gráficos tiene encima de él una serie de hileras —más planos donde poder realizar dibujos. Estos planos extras pueden ser utilizados por los duendes. En efecto, si Vd. pone un duende en el plano cero y otro en el plano uno, el duende del plano uno parecerá estar encima del duende del plano cero. Inteligente ¿verdad? Pero ¿cómo se hacen aparecer los duendes y cómo puede definirlos?

Los duendes están compuestos por 8 números binarios, un conjunto de 64 unos o ceros. Si Vd. coloca estos números como una malla de 8 × 8 tiene Vd. un diseño elemental, como se muestra en la figura 7, sobre el cual dibujar los duendes.

Ahora si pone un '1' en uno de los cuadrados de la malla con ello se fijaría una parte del duende, es decir, en la pantalla aparecería ese elemento del duende. Si Vd. pone un cero en el cuadrado, entonces ese elemento concreto del duende no aparecerá en la pantalla. Veamos esto

Fig. 7.–Malla de definición de los duendes.



con más detalle a través de un ejemplo elaborado. El siguiente programita pondrá un cuadrado en el centro de la pantalla en alta resolución. El duende aparecerá como un cuadrado relleno ya que todos los valores de las sentencias DATA son unos –los 64 elementos que configuran el duende se "encenderán".

```
10 SCREEN 2,0,0
20 FOR I=1 TO 8
30 READ B$
40 SHAPE$=SHAPE$+CHR$(VAL("&B"+B$))
50 NEXT I
60 SPRITE$(0)=SHAPE$
70 PUT SPRITE 0,(128,96),15,0
80 GOTO 80
90 DATA 11111111
100 DATA 11111111
110 DATA 11111111
120 DATA 11111111
130 DATA 11111111
140 DATA 11111111
150 DATA 11111111
160 DATA 11111111
```

SPRITE$ de la línea 60 es una variable especial del MSX-BASIC. Se utiliza expresamente para definir duendes. Lo que el bucle FOR hace es tomar, una a una, las líneas de números de las sentencias DATA e incluirlas en la variable de tipo cadena. Como seguramente habrá observado, se realiza un proceso importante de conversión en la línea 40. La función VAL, que aparece aquí, convierte los caracteres que representan números en su equivalente numérico, por ejemplo PRINT VAL

("1232") dará como resultado 1232 que es un número en simple precisión.

La línea 30 lee los números de cada sentencia DATA como una cadena de caracteres; la línea 40 añade el prefijo &B de número binario, convierte esta cadena en un valor realmente binario y a continuación convierte el número binario completo en una cadena de caracteres utilizando la función CHR$(). Cuando se ha configurado un valor para la variable de tipo cadena llamada SHAPE$ mediante la sentencia FOR, ya está lista para ser asignada a la variable especial SPRITE$. En este punto hay que darle un número al duende. Tiene Vd. 32 opciones, del 0 al 31. El número de duende se utiliza para referenciar a ese duende en particular a partir de ese momento. La línea 60 asigna el número de referencia 0 al duende que acabamos de definir.

Una vez definido el duende, querrá Vd. situarlo en la pantalla de gráficos. Esto se hace con la sentencia PUT SPRITE. Determina no sólo *dónde* aparecerá el duende, en la pantalla; también define cuál será el duende a visualizar, su color y el plano de pantalla en el que se moverá. Para el ejemplo anterior, hemos dicho que el duende 0 aparecerá en el centro de la pantalla sobre el plano cero y en color blanco. Con otras palabras, la sentencia PUT SPRITE puede describirse de la siguiente forma:

COLOCAR (PUT) un DUENDE (SPRITE) sobre el PLANO NUMERO XX en la posición (X,Y) con el color YY y el duende utilizado será el DUENDE NUMERO ZZ.

La esquina superior izquierda del duende se utiliza para referenciar dónde aparecerá en la pantalla. Es esta esquina del duende la que aparecerá situada en el centro de la pantalla, y no el centro del duende. ¡Es muy importante, que recuerde esto!

Sucede una cosa muy interesante al intentar situar a dos duendes en el mismo sitio a la vez. Los dos duendes parpadearán porque sólo puede existir un duende en un plano, en un momento dado. Lo que el chip de vídeo hace es colocar un duende en un plano. Si se encuentra con que otro duende quiere aparecer en ese plano, borra el primer duende y lo sustituye con el nuevo. El siguiente paso es intentar mover un duende por toda la pantalla.

Para mover el duende usaremos la rutina de mandos de juego, como hicimos anteriormente. Hay un duende inmóvil en el centro del plano 1. Vd. verá como si la cruz negra se moviese *sobre* la cruz blanca. Si Vd. pone la cruz negra en el plano 1 y la cruz blanca en el plano 0 podrá mover la cruz negra por debajo de la blanca.

Si observa las sentencias de datos del programa a la distancia y con el enfoque adecuado podrá distinguir borrosamente que la paterna de unos forman una cruz. Ahora el programa:

```
20 SCREEN 2,0,0
30 FOR I=1 TO 8
40 READ B$
50 SHAPE$=SHAPE$+CHR$(VAL("&B"+B$))
60 NEXT I
70 SPRITE$(0)=SHAPE$
80 SPRITE$(1)=SHAPE$
90 PUT SPRITE 0,(128,96),15,0
100 X=128:Y=92
110 DIR=STICK(0):IF DIR=0 THEN 110
120 GOSUB 150
130 PUT SPRITE 1,(X,Y),1,0
140 GOTO 110
150 IF DIR=1 THEN Y=Y-1
160 IF DIR=2 THEN Y=Y-1:X=X+1
170 IF DIR=3 THEN X=X+1
180 IF DIR=4 THEN X=X+1:Y=Y+1
190 IF DIR=5 THEN Y=Y+1
200 IF DIR=6 THEN Y=Y+1:X=X-1
210 IF DIR=7 THEN X=X-1
220 IF DIR=8 THEN X=X-1:Y=Y-1
230 IF X>256 THEN X=256
240 IF X<1 THEN X=1
250 IF Y>192 THEN Y=192
260 IF Y<1 THEN Y=1
270 RETURN
280 DATA 10000001
290 DATA 01000010
300 DATA 00100100
310 DATA 00011000
320 DATA 00011000
330 DATA 00100100
340 DATA 01000010
350 DATA 10000001
```

Estableciendo una opción del mandato SCREEN, se pueden hacer los duendes de 16 × 16 pixels. Sustituya la línea 10 del programa anterior por SCREEN 2,1,0. Con esto se agrandan las cruces del programa.

Vd. puede definir sus propios duendes de 16 X 16 sin tener que agrandar los duendes de 8 X 8. En primer lugar, hay que seleccionar una modalidad de pantalla que le permita la utilización de duendes de 16 X 16. La sentencia que realiza este truco es SCREEN 2,2,0. Si se está Vd. preguntando porqué hay que desactivar el "click" cada vez que se utiliza la sentencia SCREEN, probablemente estará de acuerdo en que el ruidito constante producido cuando se está utilizando continuamente el mandato de juego es francamente molesto.

Para definir duendes de 16 X 16, se necesitan 32 sentencias DATA. Las 16 primeras sentencias DATA definen la figura de la mitad izquierda del duende, las 16 restantes definen la mitad derecha.

Por lo demás, el procedimiento para su tratamiento y creación es similar al descrito para los duendes de 8 X 8. También puede agrandar estos duendes como si estuviesen dimensionados a 32 X 32 pixels utilizando el mandato: SCREEN, 2,3,0.

He aquí unos juegos sencillos, para darles algunas ideas. El primero utiliza un duende de 16 X 16 para representar una nave espacial, un duende de 8 X 8 para representar un misil y los blancos son filas de rectángulos rojos. El juego dura hasta que acierte en los blancos cinco veces. Es muy sencillo y proporciona una base razonable para desarrollar sus ideas sobre sus propios juegos. En el programa se han incluido algunas características especiales. Cuando el mando de juego no se está moviendo en ninguna dirección, la nave espacial se moverá por la pantalla en direcciones aleatorias.

```
10 REM JUEGO DE DEMOSTRACION
20 DEFINT A-Z
25 SCREEN 2,2:COLOR 15,15,15:CLS
30 REM INICIALIZAR VARIABLE
35 ON STRIG GOSUB 860:REM TRATAMIENTO CA
IDA MISILES
40 SC=0:REM NUMERO DE ACIERTOS
50 X=128:Y=10:REM COORDENADAS INICIALES
DE LA NAVE ESPACIAL
60 REM DIBUJAR RECTANGULOS
70 LINE(40,180)-(50,190),4,BF
80 LINE(80,180)-(90,190),4,BF
90 LINE(120,180)-(130,190),4,BF
100 LINE(160,180)-(170,190),4,BF
110 LINE(200,180)-(210,190),4,BF
115 LINE (0,96)-(256,96),1
120 REM SPRITE (DUENDE) NAVE ESPACIAL
```

```
130 FOR I=1 TO 32
140 READ A$
150 S$=S$+CHR$(VAL("&B"+A$))
160 NEXT I
170 SPRITE$(0)=S$
180 REM SPRITE (DUENDE) MISIL
190 REM  RESTORE 1260
200 FOR K=1 TO 8
210 READ B$
220 T$=T$+CHR$(VAL(("&B")+B$))
230 NEXT K
240 SPRITE$(1)=T$
250 REM POLLING A LA BARRA ESPACIADORA
260 STRIG(0) ON
270 REM POSICION DE LA NAVE ESPACIAL
280 PUT SPRITE 0,(X,Y),1,0
300 REM VER SI SE HA MOVIDO EL MANDO
310 REM SI NO, CALCULAR DIRECCION AL
AZAR
320 N=STICK(0):IF N=0 THEN N=INT(RND(1)*
8)
330 REM CALCULAR DESPLAZAMIENTOS
340 IF N=0 THEN GOTO 280
350 ON N GOSUB 760,770,780,790,800,810,8
20,830
360 REM COMPROBAR SI LOS VALORES DE X E
Y SEAN VALIDOS
370 IF X>256 THEN X=256
380 IF X<1 THEN X=1
390 IF Y>104 THEN Y=104
400 IF Y<1 THEN Y=1
410 GOTO 280
420 REM SUBRUT. PARA OBTENER DESPLAZAM.
760 Y=Y-1:RETURN
770 Y=Y-1:X=X+1:RETURN
780 X=X+1:RETURN
790 X=X+1:Y=Y+1:RETURN
800 Y=Y+1:RETURN
810 Y=Y+1:X=X-1:RETURN
820 X=X-1:RETURN
830 X=X-1:Y=Y-1:RETURN
```

```
840 REM SUBRUTINA PARA MANEJAR LA PULSA
CION DEL DISPARADOR
850 REM CAIDA DE BOMBA Y VER SI ACIERTA
EN EL BLANCO
860 V=Y+1
870 PUT SPRITE 1,(X+4,V),1,1
880 IF POINT((X+12),(V+8))=4 THEN PLAY"S
M1500L2406CDEFG":SC=SC+1:IF SC>4 THEN SC
REEN 1:COLOR 15,4,7:CLS:END ELSE :RETURN
890 IF V=192 THEN PLAY "C","F+","B":RETU
RN
900 V=V+1:GOTO 870
910 REM SENTENCIAS DATA PARA DEFINICION
DE LOS DUENDES (SPRITES)
920 REM 1. NAVE ESPACIAL
930 DATA 11110000
940 DATA 11110000
950 DATA 11000000
960 DATA 01000010
970 DATA 01000111
980 DATA 01001010
990 DATA 01111111
1000 DATA 01101011
1010 DATA 01101010
1020 DATA 01111111
1030 DATA 01001010
1040 DATA 01000111
1050 DATA 01000000
1060 DATA 11000000
1070 DATA 11110000
1080 DATA 11110000
1090 DATA 00001111
1100 DATA 00001111
1110 DATA 00000011
1120 DATA 10000010
1130 DATA 11100010
1140 DATA 01010010
1150 DATA 11111110
1160 DATA 01010110
1170 DATA 01010110
1180 DATA 11111110
1190 DATA 01010010
```

```
1200 DATA 11100010
1210 DATA 10000010
1220 DATA 00000011
1230 DATA 00001111
1240 DATA 00001111
1250 REM 2. DEFINICION DE LA BOMBA
1260 DATA 00011000
1270 DATA 11111111
1280 DATA 11111111
1290 DATA 10000001
1300 DATA 01000010
1310 DATA 00100100
1320 DATA 00111100
1330 DATA 00111100
```

El siguiente programa es un juego de aterrizaje lunar cuyo objetivo es conseguir un aterrizaje sobre tres pistas blancas de aterrizaje. Al pulsar la barra espaciadora se reduce la velocidad de descenso, pero se irá reduciendo gradualmente el combustible. Existen inconvenientes, desde luego,. Hay que evitar "manchas" móviles y estacionarias. Si Vd. toca una de ellas, la nave estallará. Del mismo modo, si se estrella contra las rocas, o se queda sin combustible, Vd. destruirá su nave. Si el alunizaje es correcto, un hombrecito salta a un cuadrado de la parte inferior de la pantalla. El programa hace uso de otro mandato ON...GOSUB similar a la sentencia ON STRIG GOSUB que ya hemos visto. Este comando le permite controlar la colisión entre duendes, en este caso, los duendes que representan las manchas colisionando con la nave espacial. Hay que ejecutar antes el mandato SPRITE ON para que el BASIC sea capaz de detectar un choque entre duendes. SPRITE OFF y SPRITE STOP son como los respectivos mandatos STRIG() ON y STRIG() OFF. A continuación se dan las líneas maestras de cómo utilizar estas sentencias en un programa:

```
10 SPRITE ON:REM COMIENZO DETECCION DE
COLISIONES
.
.
.
70 ON SPRITE GOSUB 100:REM VER CHOQUE EN
TRE SPRITES
.
.
.
```

```
100 REM SUBRUTINA TRATAM. COLISIONES
110 SPRITE OFF:REM DESACTIVAR DETECCION
DE COLISIONES
.
.
.
180 SPRITE ON:REM ACTIVAR DE NUEVO LA DE
TECCION DE COLISIONES
190 RETURN
```

Y aquí está el programa, como prometimos:

```
10 REM JUEGO DEL ALUNIZAJE
20 REM
30 SCREEN 2,0,0
40 CLEAR
50 FLAG=1
60 COL=15
70 DIR=4:K=63
80 STRIG(0) ON
90 ON STRIG GOSUB 1500
100 INC=1
110 FUEL=100
120 FFLAG=0
130 K=100
140 F=0
150 ACC=3
160 X=128:Y=10
170 COLOR 15,4,1
180 REM CREAR SPRITES
190 N=0
200 GOSUB 820
210 N=1:RESTORE 1050:OBJ$="":GOSUB 820
220 N=2:RESTORE 1140:OBJ$="":GOSUB 820
230 GOSUB 350
240 X=128:Y=10:PUT SPRITE 0,(X,Y),8,1:TL
=0:SPRITE ON
250 ON SPRITE GOSUB 1570
260 REM polling a las palancas de juego
270 SOUND 6,K:S=STICK(0):IF S=0 OR S=1 O
R S=8 THEN S=DIR
```

```
280 GOSUB 880
290 GOSUB 1230
300 GOSUB 1270
310 GOSUB 1440
320 REM
330 GOTO 270
340 REM DIBUJAR PAISAJE
350 LINE(0,160)-(24,184),1
360 LINE(24,184)-(32,144),1
370 LINE(32,144)-(40,160),1
380 LINE(40,160)-(48,152),1
390 LINE(48,152)-(56,168),1
400 LINE(56,168)-(64,152),1
410 LINE(64,152)-(56,144),1
420 LINE(56,144)-(80,144),1
430 LINE(80,144)-(72,152),1
440 LINE(72,152)-(88,160),1
450 LINE(88,160)-(96,104),1
460 LINE(96,104)-(104,128),1
470 LINE(104,128)-(120,128),1
480 LINE(120,128)-(112,144),1
490 LINE(112,144)-(128,168),1
500 LINE(128,168)-(136,152),1
510 LINE(136,152)-(144,168),1
520 LINE(144,168)-(160,136),1
530 LINE(160,136)-(160,160),1
540 LINE(160,160)-(176,120),1
550 LINE(176,120)-(184,144),1
560 LINE(184,144)-(200,128),1
570 LINE(200,128)-(216,128),1
580 LINE(216,128)-(232,144),1
590 LINE(232,144)-(236,128),1
600 LINE(236,128)-(208,96),1
610 LINE(208,96)-(224,80),1
620 LINE(224,80)-(256,136),1
630 LINE(56,142)-(80,142),15
640 LINE(104,126)-(120,126),15
650 LINE(200,126)-(216,126),15
660 REM COLOCAR OBSTACULOS ESTACIONA.
670 PUT SPRITE 10,(32,56),3,2
680 PUT SPRITE 12,(40,92),3,2
690 PUT SPRITE 13,(148,54),3,2
```

```
700 PUT SPRITE 14,(104,50),3,2
710 PUT SPRITE 15,(208,50),3,2
720 PUT SPRITE 16,(112,92),3,2
730 PUT SPRITE 17,(184,90),3,2
740 PUT SPRITE 18,(160,20),3,2
750 CIRCLE (0,0),70,8:PAINT (1,1),8
760 PAINT (128,191),1
770 LINE(200,184)-(224,192),15,B
780 LINE(208,184)-(208,192),15
790 LINE(216,184)-(216,192),15
800 RETURN
810 REM RUTINA CREACION SPRITES
820 FOR I=1 TO 8
830 READ A$
840 OBJ$=OBJ$+CHR$(VAL("&B"+A$))
850 NEXT I
860 SPRITE$(N)=OBJ$:RETURN
870 REM CALCULO DESPLAZ. MANDOS DE JUEGO
880 IF S=4 THEN 900
890 DIR = S
900 IF S=3 THEN X=X+2:RETURN
910 IF S=4 THEN X=X+1:Y=Y+ACC:RETURN
920 IF S=5 THEN Y=Y+ACC:RETURN
930 IF S=6 THEN Y=Y+ACC:X=X=1:RETURN
940 IF S=7 THEN X=X-1:Y=Y+1:RETURN
950 REM PATERNA DE NAVE ESPACIAL
960 DATA 00011000
970 DATA 00011000
980 DATA 00111100
990 DATA 00111100
1000 DATA 00111100
1010 DATA 11111111
1020 DATA 10000001
1030 DATA 10000001
1040 REM PATERNA DE HOMBRE
1050 DATA 00000000
1060 DATA 00011000
1070 DATA 00011000
1080 DATA 01111110
1090 DATA 00011000
1100 DATA 00111100
1110 DATA 00111100
```

```
1120 DATA 00000000
1130 REM PATERNA DE MANCHA
1140 DATA 00111000
1150 DATA 01111101
1160 DATA 00111110
1170 DATA 01111111
1180 DATA 11111111
1190 DATA 01111100
1200 DATA 00111110
1210 DATA 00011111
1220 REM COMPROB. RANGO Y SITUAR NAVE
ESPACIAL Y MANCHAS
1230 IF X<0 THEN X=0:RETURN
1240 IF Y<0 THEN Y=0:RETURN
1250 IF X>256 THEN X=256:RETURN
1260 IF Y>192 THEN Y=192:RETURN
1270 REM COLOCAR MANCHAS EN MOVIMIENTO
1280 PUT SPRITE 0,(X,Y),8,0
1290 PUT SPRITE 2,(F,80),3,2:F=F+1
1300 PUT SPRITE 3,(F+100,100),3,2
1310 PUT SPRITE 4,(F+10,40),3,2
1320 PUT SPRITE 5,(F+60,128),3,2
1330 PUT SPRITE 6,(F+112,116),3,2
1340 PUT SPRITE 7,(F+224,124),3,2
1350 PUT SPRITE 8,(F+72,72),3,2
1360 IF FLAG=1 THEN RETURN
1370 GOTO 1400
1380 PUT SPRITE 1,(X,Y+4),15,1
1390 PUT SPRITE 1,(X,Y+4),0,1
1400 FUEL=FUEL-2:K=K-4:IF K<0 THEN K = 6
3
1410 IF FUEL=0 THEN GOTO 1570
1420 REM RETURN
1430 REM SEE IF THE SHIP HAS LANDED YET
1440 IF (POINT(X,Y+9)=15) AND POINT(X+9,
Y+9)=15 THEN BEEP:PLAY"L20O6CDEFG":X=128
:Y=10:FFLAG=0:FLAG=1:FUEL=100:SC=SC+1
1450 IF SC=1 THEN PUT SPRITE 23,(200,184
),15,1
1460 IF SC=2 THEN PUT SPRITE 23,(208,182
),15,1
```

```
1470 IF SC=3 THEN PUT SPRITE 23,(216,184
),15,1:SC=0
1480 IF POINT(X,Y+9)=1 OR POINT(X+9,Y+9)
=1 THEN GOTO 1570
1490 RETURN 330
1500 SWAP FLAG,FFLAG:SOUND 7,5:SOUND 8,7
1510 ACC=1:IF FFLAG=0 THEN PUT SPRITE 1,
(0,0),0,1
1520 IF FFLAG=0 THEN ACC=3:SOUND 8,0
1530 RETURN
1540 END
1550 REM RUTINA DE COLISIONES
1560 TL=1
1570 BEEP
1580 FOR I=1 TO 4
1590 PUT SPRITE 0,(X,Y),1,0
1600 PUT SPRITE 0,(X,Y),15,0
1610 FOR J=1 TO 255 STEP 5:SOUND 8,9:SOU
ND 0,J:SOUND 8,0:NEXT J
1620 NEXT
1630 IF TL=1 THEN GOTO 240
1640 X=128:Y=10:FUEL=100:FLAG=1:FFLAG=0
1650 RETURN
```

Dibujar

El mandato DRAW (DIBUJAR) se utiliza para dibujar figuras utilizando el Macro Lenguaje de Gráficos (o GML). Funciona de la misma forma que el MML que vimos en el capítulo anterior. Las instrucciones de dibujo se presentan como cadenas de caracteres conteniendo mandatos del GML.

Intente el siguiente ejemplo:

```
10 SCREEN 2
20 DRAW "R100D100L100U100"
30 GOTO 30
```

Como puede ver, este programa dibuja un cuadrado en la pantalla. El lugar en el que se posiciona el cuadrado depende de si Vd. ha utilizado o no alguna de las modalidades gráficas con anterioridad. Si Vd. introduce este programa nada más conectar la máquina, el cuadrado se dibujará en la esquina superior izquierda de la pantalla. Sin embargo, si ya

utilizó antes alguna de las pantallas de gráficos entonces la esquina superior izquierda del cuadrado se posicionará en el último pixel direccionado.

Lo que el ordenador ha hecho con el mandato DRAW es dibujar una serie de líneas; primero 100 pixels hacia la derecha después 100 pixels hacia abajo retrocediendo 100 pixels para terminar en el punto donde se empezó la primera línea y dibujando así el cuadrado —¡tan sencillo como esto! Ahora sustituya la línea 20 con:

```
20 DRAW "D100F100E100H100G100"
```

Con esto, se dibuja primero una línea vertical de 100 puntos hacia abajo luego, diagonalmente hacia abajo y a la derecha, hacia arriba y a la derecha, hacia arriba y a la izquierda y por último, hacia abajo, y a la izquierda, utilizando cuatro elementos más de la notación GML, llamados F, E, H y G. Esta es la lista completa:

U <n> Movimiento hacia arriba
D <n> Movimiento hacia abajo
L <n> Movimiento hacia la izquierda
R <n> Movimiento hacia la derecha
E <n> Movimiento diagonal arriba y derecha
F <n> Movimiento diagonal abajo y derecha
G <n> Movimiento diagonal abajo e izquierda
H <n> Movimiento diagonal arriba e izquierda

Para todos estos mandatos, el movimiento empieza en el último punto referenciado. Esto significa que el movimiento se inicia en la esquina superior izquierda de la pantalla y después sigue a partir del punto en el que el mandato anterior terminó su dibujo. <n> indica la distancia que se va a recorrer, de manera que R100 significa moverse 100 pixels hacia la derecha, F100 significa moverse 100 pixels hacia abajo y 100 pixels hacia la derecha y así sucesivamente.

Un sistema para empezar a dibujar en un punto distinto de la esquina superior izquierda es utilizar:

M <x,y>

donde x representa un movimiento horizontal e y un movimiento vertical. Inténtelo, cambiando la línea 20 del programa, otra vez, a:

```
20 DRAW "M100,10R100D100L100U100"
```

Como puede ver, el punto a partir del cual se empieza a dibujar se ha movido 100 pixels a lo largo de la pantalla y 10 pixels hacia abajo.

A cualquiera de los comandos anteriores se les puede añadir el prefijo B o N. Con B se produce el movimiento pero no hay trazado de puntos, N produce movimiento pero retorna a la posición original de comienzo cuando haya terminado. Otros comandos disponibles son:

A<n> le permite establecer el ángulo con el que se va a dibujar una línea. n puede variar entre 0 y 3 donde 0 es 0 grados, 1 es 90, 2 es 180 y 3 es 270 grados.

```
    0
1-- + --3
    2
```

C<n> establece el color de la línea que se va a dibujar y al igual que en el mandato COLOR, n puede variar de 0 a 15.

S<n> establece un factor de escala que puede ser utilizado para modificar la distancia recorrida utilizando los mandatos U, D, L, R, E F, G, H y M.

X<variable tipo cadena> le permite ejecutar un mandato DRAW desde dentro de otro mandato DRAW. Esto puede serle muy útil ya que, por ejemplo, Vd. puede definir parte del objeto que está dibujando, como algo independiente del objeto, o puede ejecutar mandatos DRAW que tengan más de 255 caracteres de largo.

El programa "Caja de Pinturas"

El siguiente programa se llama Caja de Pinturas y es un sencillo programa de dibujos, de uso general. La pantalla de gráficos se divide en dos áreas: un área de dibujo y un área de selección del color. Se utiliza un mando de juego (o las teclas del cursor) para controlar un cursor.

El área de selección del color contiene una caja de pinturas que da nombre al programa. Es una paleta de catorce colores pudiéndose seleccionar cualquiera de ellos, posicionando el cursor encima del color deseado y pulsando la barra espaciadora. Los colores que no están disponibles en la paleta son el 0 —transparente, y el 15 —blanco.

Los instrumentos de dibujo que puede Vd. utilizar son: un pincel, un tira-líneas (para dibujar rectas), un compás (para dibujar círculos), una figura rectangular (para dibujar rectángulos) y un borrador. También puede elegir entre cuatro composiciones "para rellenar" —un bote de aerosol, líneas verticales, líneas horizontales y una paterna de puntos,

así como la opción PAINT estándar. El cursor toma la figura del instrumento de dibujo seleccionado (o la paterna de composición) mientras se está moviendo dentro de los límites del área de dibujo. El término icon (icono, representación) se utiliza para definir estas diferentes figuras del cursor –la imagen del cursor representa la función actualmente seleccionada.

Se pueden seleccionar varias funciones (20 en total) desde el teclado. A continuación damos la relación completa de todas las letras de mandato (observe que todas están en mayúsculas). Siempre que sea posible se debe intentar que la letra de mandato coincida con la función del mandato –¡aunque al cabo de un tiempo nos hayamos quedado sin letras!

Utilizando el pincel

D selecciona el pincel y lo pone "hacia abajo". Moviendo el cursor hacia abajo se dibuja una línea. Inicialmente el pincel dibujará una línea muy estrecha. Pulsando D, de nuevo, podrá pintar con toda la anchura del pincel.

U "levanta" el pincel con lo que puede moverlo sin dejar ninguna traza.

Utilizando el bote de aerosol

A selecciona el bote de aerosol (¡pintura de spray!).

BARRA ESPACIADORA pinta con el spray del aerosol.

Utilizando el compás para dibujar un círculo

C selecciona el compás y define la posición actual como el centro del círculo que se va a dibujar.

R define el radio del círculo como la línea comprendida entre el punto donde estaba el cursor cuando Vd. pulsó C y la posición actual –a continuación se dibuja el círculo.

Utilizando el tira-líneas para dibujar una recta

L selecciona el tiralíneas y define la posición actual del cursor como el punto inicial de una recta.

E define el punto final de una recta —y dibuja la recta.

Utilizando el rectángulo para dibujar un rectángulo

B selecciona el rectángulo y define la posición actual del cursor como la posición inicial para una de las esquinas del rectángulo que se va a dibujar.

X define la esquina diagonalmente opuesta del rectángulo, y lo dibuja.

Pintando un rectángulo o cualquier área cerrada

P pinta todo el área cerrada dentro de la cual se ha colocado el cursor. Observe que si el área no es cerrada, se pintará toda la pantalla con color actual.

Sombreados locales utilizando cuadrados de composición/paterna

Al usar uno de estos mandatos se selecciona un pequeño cuadrado compuesto. Pulsando la barra espaciadora se dibuja este cuadrado en la posición actual con la composición/paterna seleccionada.

V líneas verticales —selecciona una paterna de relleno de líneas verticales.

Y selecciona una paterna de relleno de líneas horizontales.

K selecciona una paterna de relleno de puntos.

Utilizando el borrador

W selecciona el borrador. Al mover el cursor, borra (pintando con pixels blancos).

Otros mandatos

Q selecciona el color blanco (mientras se está en el área de dibujo).

T cambia el cursor —alterna el color del cursor de blanco a negro o viceversa.

H home —salta al área de pantalla de selección del color.

S screen —salta al centro de la sección de dibujo.

Z ¡opción cero! — pone en blanco el área de la pantalla de dibujo. Para evitar que accidentalmente se pueda destruir su obra maestra, hay que utilizar el comando Z dos veces antes de que produzca ningún efecto.

F salida del programa. Al igual que el mandato Z, también hay que pulsarlo dos veces antes de que termine el programa.

Este programa utiliza ampliamente los duendes para definir distintas figuras del cursor, hasta un total de diez. En todos los casos, el punto de referencia para todas las figuras del cursor es la esquina superior izquierda —el punto inicial para un rectángulo quedará definido a partir de donde esté posicionada la esquina superior izquierda del cursor.

Se utiliza una función definida por el usuario para calcular el radio de un círculo mediante el Teorema de Pitágoras.

Cuando el cursor se mueve dentro del área de selección del color, se convierte en una flecha. Cuando pasa al área de dibujo, el cursor toma de nuevo la figura del último instrumento de dibujo utilizado. El instrumento de dibujo seleccionado por defecto es el pincel.

También se realizan en el programa, algunas comprobaciones para asegurarse de que los objetos "sólo" se están dibujando dentro del área de la pantalla dedicada al dibujo. En la mayoría de los casos, las letras de comandos sólo son aceptadas mientras el cursor se encuentra dentro de los límites del área de dibujo.

Cuando se está pintando un área de la pantalla, si el cursor permanece aunque sólo sea encima de un único pixel del mismo color que con el de pintura seleccionado, Vd. no verá nada de lo que se está pintando. Por eso, antes de empezar a pintar, retire el cursor y sitúelo en los pixels de diferente color.

Cuando se está rellenando un área, el programa sólo reconocerá como fronteras, las líneas que tengan el mismo color actual. Por lo tanto, debe Vd. asegurarse de que el borde de la figura que va a pintar es del mismo color que el color actualmente seleccionado. Para evitar que el programa rellene áreas situadas fuera del área de dibujo, los bordes de este área se dibujan automáticamente con el color actual antes de empezar a pintar cualquier área. Utilizando P (Paint = Pintar) antes de dibujar cualquier cosa, se cambia todo el área de dibujo al color seleccionado, de esta forma Vd. dispone de "telas" de varios colores además del blanco.

```
10 REM CAJA DE PINTURAS
20 DEFINT A-Z
30 REM DESPLAZAMIENTOS MANDOS DE JUEGO
40 DIM Z(8,2)
50 FOR I= 1 TO 8
60 FOR J= 1 TO 2
70 READ A:Z(I,J)=A
80 NEXT J
90 NEXT I
100 DATA 0,-1,1,-1,1,0,1,1,0,1,-1,1,-1,0
,-1,-1
110 SCREEN 2,0,0
120 COLOR15,15,15:CLS
130 STRIG(0) ON
140 REM DEFINIR FUNCION DE PITAGORAS
150 DEFFNC(M,N,O,P)=SQR((ABS((M-O)^2))+(
ABS((N-P)^2)))
160 F=1:M=2:N=1:Q=1:P1=0:P2=1
170 ON STRIG GOSUB 1400
180 X=40:Y=172
190 GOSUB 320:REM CREAR PANTALLA
200 GOSUB 410:REM CREAR LOS SPRITES
210 W=15
220 REM SITUAR CURSOR
230 C5=1
240 PUT SPRITE 1,(X,Y),Q,N
250 IF N=3 THEN PSET(X,Y),15
260 IF X>57 AND FLG=1 AND N=2 THEN GOSUB
 2090
270 IF STICK(0)=0 THEN GOTO 290
280 GOSUB 1470
290 B$=INKEY$:IF B$= "" THEN GOTO 240
300 GOSUB 1560
305 GOTO 240
310 REM RUTINA DIBUJAR PANTALLA
320 C=1
330 FOR I=8 TO 32 STEP 24
340 FOR J=8 TO 152 STEP 24
350 LINE (I,J)-(I+16,J+16),C,BF:C=C+1
360 NEXT J
370 NEXT I
380 LINE (56,8)-(248,184),1,B
```

```
390 RETURN
400 REM RUTINA CREAR SPRITE
410 FOR I=1 TO 10
420 S$=""
430 FOR I=1 TO 8
440 READ A$:S$=S$+CHR$(VAL("&B"+A$))
450 NEXT I
460 SPRITE$(I)=S$
470 NEXT I
480 RETURN
490 REM SPRITE FLECHA
500 DATA 11110000
510 DATA 11000000
520 DATA 10100000
530 DATA 10010000
540 DATA 00001000
550 DATA 00000100
560 DATA 00000010
570 DATA 00000001
580 REM SPRITE PINCEL
590 DATA 11110000
600 DATA 11110000
610 DATA 11110000
620 DATA 00000000
630 DATA 11110000
640 DATA 01100000
650 DATA 01100000
660 DATA 01100000
670 REM SPRITE BORRADOR
680 DATA 11110000
690 DATA 10011000
700 DATA 10010100
710 DATA 11110010
720 DATA 01000001
730 DATA 00100001
740 DATA 00010001
750 DATA 00001111
760 REM SPRITE COMPAS
770 DATA 11000000
780 DATA 00100000
790 DATA 00010010
800 DATA 00001111
```

```
810 DATA 00001111
820 DATA 00010010
830 DATA 00100000
840 DATA 11000000
850 REM BOTE DE AEROSOL
860 DATA 11100000
870 DATA 00111110
880 DATA 00100010
890 DATA 11111010
900 DATA 10001000
910 DATA 10001000
920 DATA 10001000
930 DATA 11111000
940 REM SPRITE LINEA
950 DATA 10000000
960 DATA 10000000
970 DATA 10000000
980 DATA 10000000
990 DATA 10000000
1000 DATA 10000000
1010 DATA 10000000
1020 DATA 10000000
1030 REM SPRITE RECTANGULO
1040 DATA 11111111
1050 DATA 10000001
1060 DATA 10000001
1070 DATA 10000001
1080 DATA 10000001
1090 DATA 10000001
1100 DATA 10000001
1110 DATA 11111111
1120 REM BARRA HORIZONTAL
1130 DATA 00000000
1140 DATA 11111111
1150 DATA 00000000
1160 DATA 11111111
1170 DATA 00000000
1180 DATA 11111111
1190 DATA 00000000
1200 DATA 11111111
1210 REM BARRA VERTICAL
1220 DATA 10101010
```

```
1230 DATA 10101010
1240 DATA 10101010
1250 DATA 10101010
1260 DATA 10101010
1270 DATA 10101010
1280 DATA 10101010
1290 DATA 10101010
1300 REM PUNTEAR CURSOR
1310 DATA 10101010
1320 DATA 01010101
1330 DATA 10101010
1340 DATA 01010101
1350 DATA 10101010
1360 DATA 01010101
1370 DATA 10101010
1380 DATA 01010101
1390 REM RUTINA INTERRUPCION BARRA ESPAC
1400 IF X>57 AND N=5 THEN GOSUB 1930:RET
URN
1410 IF X>57 AND N=8 THEN V1=1:V2=2:GOSU
B 2000:RETURN
1420 IF X>57 AND N=9 THEN V1=2:V2=1:GOSU
B 2000:RETURN
1430 IF X>57 AND N=10 THEN V1=2:V2=2:GOS
UB 2000:RETURN
1440 IF X>57 THEN RETURN
1450 IF POINT(X,Y)=15 THEN PLAY "L1402C"
:RETURN
1460 C5=POINT(X,Y):PLAY "L2405C06C":RETU
RN
1470 X=X+Z(STICK(0),1):Y=Y+Z(STICK(0),2)
1480 IF X>246 THEN X=246
1490 IF Y>183 THEN Y=183
1500 IF Y<9 THEN Y=9
1510 IF X<9 THEN X=9
1520 IF X<56 THEN N=1
1530 IF X>56 THEN N=M:F=0
1540 RETURN
1550 REM RUTINA LETRA COMANDO
1560 IF (ASC(B$)<65) THEN RETURN
1570 IF B$="F" THEN BEEP:E1=E1+1:IF E1=2
 THEN SCREEN 1:COLOR 15,5,4:END
```

```
1580 IF X<56 AND B$="S" THEN X=152:Y=96:
N=M:RETURN
1590 IF B$="T" THEN SWAP Q,W:RETURN
1600 IF X<56 THEN RETURN
1610 CL=CW
1620 IF B$="Q" THEN C5=15:RETURN
1630 IF B$="U" THEN FLG=0:N=2:M=2:SWAP P
1,P2:RETURN
1640 IF B$="D" THEN FLG=1:N=2:M=2:SWAP P
1,P2:RETURN
1650 IF B$="H" THEN X=40:Y=176:N=1:RETUR
N
1660 IF B$="R" AND F2=1 THEN GOSUB 1810:
F2=0:RETURN
1670 IF B$="P" THEN LINE(56,8)-(248,184)
,C5,B:PAINT (X,Y),C5:RETURN
1680 IF B$="Z" THEN BEEP:Z=Z+1:IF Z>1 TH
EN LINE(56,8)-(248,184),15,BF:LINE (56,8
)-(248,184),1,B:Z=0:RETURN
1690 IF B$="C" THEN C1=X:C2=Y:M=4:N=4:F2
=1:RETURN
1700 IF B$="Y" THEN M=8:N=8:RETURN
1710 IF B$="V" THEN M=9:N=9:RETURN
1720 IF B$="K" THEN M=10:N=10:RETURN
1730 IF B$="W" THEN N=3:M=3:CW=CL:CL=15:
FLG=1:RETURN
1740 IF B$="L" THEN L=1:L1=X:L2=Y:N=6:M=
6:RETURN
1750 IF B$="B" THEN B=1:B1=X:B2=Y:N=7:M=
7:RETURN
1760 IF B$="X" AND B=1 THEN LINE(B1,B2)-
(X,Y),C5,B:B=0:RETURN
1770 IF B$="E" AND L=1 THEN LINE(L1,L2)-
(X,Y),C5:L=0:RETURN
1780 IF B$="A" THEN N=5:M=5
1790 RETURN
1800 REM CALCULO DEL RADIO
1810 R = FNC(X,Y,C1,C2)
1820 D1=C1:D2=C2
1830 GOSUB 1860
1840 CIRCLE (C1,C2),R,C5,,,8/7
1850 RETURN
```

```
1860 REM CALCULO DEL RANGO DE CIRCULOS
1870 IF D1+R>246 THEN BEEP:RETURN
1880 IF D1-R<57 THEN BEEP:RETURN
1890 IF D2+R>183 THEN BEEP:RETURN
1900 IF D2-R<9 THEN BEEP:RETURN
1910 RETURN 1850
1920 REM RUTINA DIBUJO AEROSOL
1930 IF (X-8<57)OR(Y-8<9) THEN RETURN
1940 RV=5*RND(1):IF RV<2 THEN 1940
1950 FOR I=1 TO RV
1960 PSET(X-(8*RND(1)),Y-(8*RND(1))),C5
1970 NEXT I
1980 RETURN
1990 REM RUTINA HOR. VERT. Y PUNTEADO
2000 IF (X+8)>246 OR (Y+8)>183 THEN BEEP
:RETURN
2010 FOR I=X TO X+7 STEP V1
2020 FOR J=Y+1 TO Y+7 STEP V2
2030 PSET(I,J),C5
2040 NEXT J
2050 NEXT I
2060 RETURN
2070 REM PINCEL PINTANDO
2080 IF P1=1 THEN PSET(X,Y),C5:RETURN
2090 WL=3
2100 IF X+WL>246 THEN WL=WL-1:GOTO 2120
2110 FOR I=0 TO WL
2120 PSET(X+I,Y),C5
2130 NEXT I
2140 RETURN
```

Mejoras sugeridas para el programa Caja de Pinturas

Utilizando el mandato C (Circle = Círculo), sólo se permite el dibujo de círculos completos, dentro del área de dibujo.

Una forma, no muy elegante, pero efectiva, de cortar los círculos en los bordes del área de dibujo es volver a dibujar todo *fuera* del límite después de que se ha dibujado un círculo. Esto se consigue con la siguiente subrutina y otras modificaciones sencillas:

```
3000 REM RUTINA DE ENMASCARAR
3010 LINE (0,0)-(56,192),15,BF
3020 LINE (56,0)-(256,8),15,BF
3030 LINE (56,184)-(256,192),15,BF
3040 LINE (248,0)-(256,192),15,BF
3050 GOSUB 320
3060 RETURN
```

La rutina de verificación del rango de los círculos (líneas 1860-9000) quedará modificada de esta forma:

```
1860 REM CALCULO DEL RANGO DE CIRCULOS
1870 IF D1+R>246 THEN FC=1:RETURN
1880 IF D1-R<57 THEN FC=1:RETURN
1890 IF D2+R>183 THEN FC=1:RETURN
1900 IF D2-R<9 THEN FC=1:RETURN
```

La línea 1910 quedará borrada. La última alteración consiste en la inclusión de la línea 1845 que llama a la subrutina:

```
1845 IF FC = 1 THEN GOSUB 3000: FC = 0
```

Aunque el hecho de volver a dibujar la pantalla comporte el desarrollo de mucho trabajo relativo a gráficos, la subrutina se ejecuta muy rápidamente.

Otra mejora podría ser la verificación de errores —en lugar de limitarse a emitir un "beep" el programa puede indicar qué es lo que ha hecho mal, de una forma visual. Se puede definir una serie de duendes de error, uno para cada tipo de error. Se puede visualizar el duende adecuado en la esquina inferior izquierda de la pantalla cuando Vd. haya cometido una equivocación. Por ejemplo, si se intenta pintar en un área de la pantalla cuando el cursor está posicionado sobre un punto del mismo color, se puede visualizar un símbolo en forma de flecha apuntando al sitio específico, indicando que hay que mover el cursor.

Se puede introducir una opción de pintura, más avanzada, permitiendo la pintura de círculos y rectángulos utilizando líneas verticales, horizontales o puntos. Mientras que esto es muy sencillo en el caso de los rectángulos, los círculos presentan mayores dificultades. Aquí sugerimos un método para rellenar rectángulos. Primero, modifique la línea 1760:

1760 IF B$ = "X" AND B = 1 THEN LINE (B1, B2) – (X, Y), C5, B: B = 0: GOSUB 4000: RETURN

```
4000 C$=INKEY$: IF C$="" THEN 4000
4010 IF ASC(C$)<48 OR ASC(C$)>51 THEN BE
EP:RETURN
4020 IF C$ = "0" THEN RETURN:REM NO SE
DESEA COMPOSICION
4030 IF C$ = "1" THEN V1=1:V2=2:GOSUB 50
00:RETURN
4040 IF C$ = "2" THEN V1=2:V2=1:GOSUB 50
00:RETURN
4050 IF C$ = "3" THEN V1=2:V2=2:GOSUB 50
00:RETURN
4060 RETURN
.
.
.
5000 IF B1>X THEN V1=-V1
5010 IF B2>Y THEN V2=-V2
5020 FOR I= B1 TO X STEP V1
5030 FOR J= B2 TO Y STEP V2
5040 PSET (I,J),C5
5050 NEXT J
5060 NEXT I
5070 RETURN
```

Una vez dado el mandato X, el programa esperará que Vd. teclee un número comprendido entre 0 y 3. Estos valores corresponden a:

0 No pintar el rectángulo.
1 Rellenar el rectángulo con líneas horizontales.
2 Rellenar el rectángulo con líneas verticales.
3 Rellenar el rectángulo con composición de puntos.

Las líneas 5020-5070 de la subrutina son prácticamente las mismas de otra rutina del programa (2010-2060). Para evitar la duplicación de estas líneas, intente alterar las líneas que llaman a la subrutina (líneas 1410, 1420 y 1430) y el comienzo de la subrutina en la línea 4000. Deberá mantener las líneas 5000 y 5010 de la nueva subrutina.

Rellenar círculos de esta manera causa muchos problemas en lo que están involucradas altas matemáticas, de forma que si se encuentra con suficientes ánimos – ¡empiece a trabajar en el tema!

Sin duda alguna habrá observado que este programa se puede ampliar enormemente. Esperamos que con éste tenga Vd. bastante para empezar. Dado que es el último programa (¡y el más largo!) del libro, parece adecuado terminar en este punto.

Apéndice A

Funciones del MSX-BASIC

ABS(X). Devuelve el valor absoluto de la expresión X.

ASC(X$). Devuelve un valor que es el código ASCII del primer carácter de la cadena X$. Si X$ es una cadena vacía, se devuelve un error de "Illegal function call".

ATN(X). Devuelve el arco tangente de X en radianes. El resultado es un valor comprendido entre –pi/2 y pi/2. La expresión puede ser de cualquier tipo numérico. Esta función se realiza siempre en doble precisión.

BIN$(X). Devuelve una cadena que es el valor binario de un número decimal. X debe ser una expresión numérica comprendida entre –32768 y 65535. Si X es negativo se utiliza la forma del complemento a dos –BIN$(–X) es lo mismo que BIN$(65535 – X).

CDBL(X). Convierte X en un número en doble precisión.

CHR$(X). Devuelve una cadena que representa el código ASCII de X.

CINT(X). Convierte X en un número entero. Si X se sale del entorno –32768 a 32767 se produce un error de "Overflow".

COS(X). Devuelve el coseno de X en radianes. Se calcula en doble precisión.

CSNG(X). Convierte X en un número en simple precisión.

CSRLIN. Devuelve la coordenada vertical (columna) del cursor.

ERL/ERR. Variables de error. Cuando se detecta un error, ERR tiene el valor del código de error y ERL contiene el número de línea en la que se ha detectado el error. Si la sentencia causante del error era una sentencia en modo directo, ERL contendrá 65535.

EXP(X). Retorna el valor de e elevado a X.

FIX(X). Retorna la parte entera de X (fracción truncada).

FRE(" "). Retorna el número de bytes libres de memoria que se pueden utilizar para los programas BASIC, variables, etc.

HEX$(X). Retorna una cadena que representa el valor hexadecimal de X. X puede valer entre –32768 y 65535.

INKEY$. Devuelve un único carácter leído desde el teclado o una cadena vacía.

INPUT$(X). Devuelve una cadena de X caracteres de largo, leídos desde el teclado.

INSTR([X,] A$, B$). Busca la primera ocurrencia de la cadena B$ en A$ y devuelve la posición de A$ en la que se produce la coincidencia. Se puede utilizar el desplazamiento X para fijar la posición inicial de la búsqueda. El rango de X variará entre 0 y 255.

INT(X). Devuelve el mayor entero < = X.

LEFT$(A$, X). Devuelve una cadena compuesta por los X caracteres más a la izquierda de la cadena A$. El rango de X variará entre 0 y 255.

LEN(X$). Devuelve el número de caracteres de X$. Se cuentan todos los caracteres no imprimibles y los espacios.

LOG(X). Devuelve el logaritmo natural de X. X debe ser mayor que cero.

LPOS(X). Devuelve la posición actual de la cabeza impresora de la impresora de líneas, dentro del buffer de impresión. Sin embargo, esta función no da necesariamente la posición actual de la cabeza impresora. X es un argumento falso.

MID$(A$, X[,Y]). Devuelve una cadena de Y caracteres de longitud desde A$ comenzando con el carácter X-avo. El rango, tanto de X como de Y, estará comprendido entre 1 y 255. Si se omite el parámetro Y, o si hay menos de Y caracteres a la derecha del carácter X-avo, se devuelven todos los caracteres que hay a la derecha.

OCT$(X). Devuelve una cadena representando el valor octal del argumento decimal X. X debe ser una expresión numérica comprendida entre −32768 y 65535.

PEEK(X). Devuelve un byte (un entero del rango 0 a 255) leído de la posición X de memoria. X debe estar comprendida entre −32768 y 65535.

POS(X). Devuelve la posición horizontal actual del cursor. La posición más a la izquierda es la 0. X es un argumento falso.

RIGHT$(A$, X). Devuelve los X caracteres más a la derecha de la cadena A$. Esta función se utiliza igual que LEFT$.

RND(X). Retorna un número aleatorio entre 0 y 1. Cada vez que se ejecuta el trabajo se genera la misma secuencia de números aleatorios.

SGN(X). Devuelve 1 (para X > 0), 0 (para X = 0), y −1 (para X < 0).

SIN(X). Devuelve el seno de X en radianes. SIN(X) se calcula en doble precisión.

SPACE$(X). Devuelve la cadena de espacios de longitud X. X debe estar comprendida entre 0 y 255.

SPC(X). Visualiza X espacios en pantalla. SPC sólo puede ser utilizada con las sentencias PRINT y LPRINT. X valdrá entre 0 y 255.

SQR(X). Devuelve la raíz cuadrada de X. X debe ser > = 0.

STR$(X). Devuelve una representación tipo cadena del valor de X.

STRING$(X, Y) y **STRING$(X, A$).** Devuelve una cadena de X caracteres de longitud. Todos los caracteres corresponden al código ASCII de Y o al primer carácter de la cadena A$.

TAB(X). Imprime espacios hasta la posición X de la consola. Si la posición de impresión actual está más allá de la posición X, TAB no hace nada. X varía entre 0 y 255. TAB sólo puede ser utilizada con las sentencias PRINT y LPRINT.

TAN(X). Devuelve la tangente de X en radianes. TAN(X) se calcula en doble precisión.

USR[<dígito>](X). Llama a la rutina de usuario en lenguaje ensamblador con el argumento X. <dígito> varía entre 0 y 9 y corresponde al dígito asignado a esa rutina en una sentencia DEFUSR anterior. Si se omite <dígito>, se asume USR(0).

VAL(X$). Devuelve el valor numérico de una cadena X$. La función VAL quita de la cadena X$ los blancos de cabecera, los tabuladores y los alimentadores de línea.

VARPTR(<nombre de variable>) y **VARPTR(£<número de fichero>).** Devuelve la dirección del primer byte de datos identificados con <nombre de variable>. Antes de la ejecución de VARPTR, hay que asignar un valor a <nombre de variable>, de lo contrario se produce el error "Illegal function call". El valor retornado será un entero comprendido entre –32768 y 32767. Si se devuelve una dirección negativa, sumando este valor a 65536 se obtiene la dirección real.

Apéndice B

Códigos de error y mensajes

1. **NEXT without FOR.** Ha comenzado un bucle FOR...NEXT sin una sentencia FOR. Una variable de la sentencia NEXT no corresponde a ninguna variable de la sentencia FOR previamente ejecutada y sin emparejar.
2. **Syntax error.** Una línea contiene alguna secuencia de caracteres incorrecta (como paréntesis que no se han cerrado, un mandato o sentencia con faltas de ortografía, puntuación incorrecta, etc.).
3. **RETURN without GOSUB.** Se ha encontrado una sentencia RETURN sin la sentencia previa GOSUB.
4. **Out of DATA.** Una sentencia READ se está ejecutando y no hay en el programa más sentencias DATA con datos sin leer.
5. **Illegal function call.** Se ha pasado a una función matemática, o de tipo cadena, un parámetro que no pertenece al rango permitido. Este error FC puede producirse también como resultado de:
   1. Un subíndice negativo o excesivamente grande.
   2. Un argumento negativo o cero en la función LOG.
   3. Un argumento negativo para SQR.
   4. Un argumento inadecuado para MID$, LEFT$, RIGHT$, INP, OUT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR$ u ON...GOTO.
6. **Overflow.** El resultado de un cálculo es demasiado grande para ser representado en el formato numérico BASIC.
7. **Out of memory.** Un programa es demasiado grande, tiene demasiados ficheros, tiene demasiados bucles o GOSUBs, demasiadas variables o expresiones que son demasiado complejas.
8. **Undefined line number.** Una referencia de línea en una sentencia GOTO, GOSUB, IF...THEN...ELSE se refiere a una línea no existente.
9. **Subscript out of range.** Se ha referenciado a un elemento de una matriz con un subíndice que sobrepasa las dimensiones de la matriz, o con un número incorrecto de subíndices.
10. **Redimensioned array.** Se han dado dos sentencias DIM para un mismo array o se ha dado una sentencia DIM para un array después de haberse establecido para dicho array la dimensión, por defecto, de 10.

11. **Division by zero.** En una expresión se intenta dividir por cero o elevar cero a un exponente negativo.

12. **Illegal direct.** Una sentencia, que no es válida en modalidad directa, se intenta introducir como un mandato en modalidad directa.

13. **Type mismatch.** A un nombre de variable de tipo cadena se le ha asignado un valor numérico o viceversa; una función que espera un argumento numérico recibe un argumento tipo cadena o viceversa.

14. **Out of string space.** Las variables de tipo cadena han provocado que el MSX-BASIC sobrepase la cantidad de memoria disponible. El MSX-BASIC asigna el espacio para las cadenas dinámicamente, hasta que se agota la memoria.

15. **String too long.** Se ha intentado crear una cadena de más de 255 caracteres de longitud.

16. **String formula too complex.** Una expresión de tipo cadena es demasiado larga o demasiado compleja. La expresión deberá partirse en expresiones más pequeñas.

17. **Can't continue.**Se ha intentado continuar la ejecución de un programa que:

    1. se ha detenido debido a un error;
    2. se ha modificado durante un alto en la ejecución; o
    3. no existe.

18. **Undefined user function.** Se ha llamado a una función FN antes de definirla con la sentencia DEF FN.

19. **Device I/O error.** Se ha producido un error de entrada/salida en una operación con la cinta, la impresora o el CRT. Es un error grave, es decir, el BASIC no puede recuperarse después de este error.

20. **Verify error.** El programa actual es diferente del programa salvado en cinta.

21. **No RESUME.** Se está ejecutando una rutina de detección de errores que no tiene una sentencia RESUME.

22. **RESUME without error.** Se ha encontrado una sentencia RESUME sin que se haya ejecutado ninguna rutina de detección de error.

23. **Unprintable error.** No existe mensaje de error para la condición de error que se ha producido. Normalmente se produce por una sentencia ERROR con un código de error sin definir.

24. **Missing operand.** Una expresión contiene un operador sin indicar ningún operando a continuación.

25. **Line buffer overflow.** Se ha introducido una línea con demasiados caracteres.

26-49. **Unprintable Errors.** No existe definición para estos códigos. Están reservados para futuras expansiones del BASIC.

50. **FIELD overflow.** Una sentencia FIELD intenta asignar más bytes de los que se especificaron para la longitud de registro de un fichero random en la sentencia

OPEN, o se ha detectado el final del buffer de FIELD mientras se está haciendo una operación de entrada/salida secuencial (PRINT#, INPUT#) a un fichero random.

51. **Internal error**. Ha ocurrido un mal funcionamiento –un especialista deberá revisar su máquina.

52. **Bad file number**. Una sentencia o mandato hace referencia a un fichero con un número de fichero que no está abierto (OPEN) o que no está en el rango de los números de ficheros especificados por la sentencia MAXFILE.

53. **File not found**. Una sentencia LOAD, KILL u OPEN hace referencia a un fichero que no existe en memoria.

54. **File already open**. Se intenta ejecutar un OPEN en modo salida secuencial para un fichero que ya está abierto; o se da un KILL para un fichero que está abierto.

55. **Input past end**. Se ejecuta una sentencia INPUT después de haber leído con INPUT todos los datos del fichero, o para un fichero nulo (vacío). Para evitar este error, utilice la función EOF para detectar el fin de fichero.

56. **Bad file name**. Se utiliza una forma no válida para el nombre de fichero con LOAD, SAVE, KILL, NAME, etc.

57. **Direct Statement in file**. Se encuentra una sentencia directa mientras se está cargando (LOAD) un fichero de formato ASCII. Se termina el LOAD.

58. **Sequential I/O only**. Se intenta ejecutar una sentencia de acceso directo con un fichero secuencial.

59. **File not OPEN**. El fichero especificado en una sentencia PRINT#, INPUT#, etcétera, no se ha abierto (OPEN).

60-255. **Unprintable Errors**. No existe definición para estos códigos. Los usuarios pueden asignar sus propios códigos de errores empezando con los valores más altos.

Apéndice C

Teclas de control para el editor de pantalla completa

TECLA DE CONTROL	TECLA ESPECIAL	FUNCION
A		Ignorada
*B		Mover cursor al comienzo de palabra anterior
*C		Cancelar cuando MSX-BASIX está esperando datos de entrada
*D		Ignorada
*E		Truncar línea (borra texto hasta el final de la línea lógica)
*F		Mover cursor al comienzo de palabra siguiente
*G		Beep
H	Retroceso	Retroceso, borra los caracteres según se retrocede
I	Tab	Tabula (se pasa al siguiente tabulador)
*J		Alimentación de línea
*K	Home	Mover cursor a posición de Home (0, 0)
*L	CLS	Limpiar la pantalla
*M	Return	Retorno de carro (entrada de la línea lógica actual)
*N		Añadir al final de línea
*O		Ignorada
*P		Ignorada
*Q		Ignorada
*R	INS	Alternar modalidad insertar/sobreescribir
*S		Ignorada
*T		Ignorada
*U		Limpiar línea lógica
*V		Ignorada
*W		Ignorada
*X	Select	Ignorada
*Y		Ignorada
*Z		Ignorada
[	ESC	Ignorada
*Y	Flecha derecha	Cursor hacia la derecha (Y es signo del Yen)

*]	Flecha izquierda	Cursor hacia la izquierda
*^	Flecha arriba	Cursor hacia arriba
*_	Flecha abajo	Cursor hacia abajo
***DEL**	DEL	Borra carácter en el que está el cursor

Todas las teclas señaladas con asterisco (*) cancelan la modalidad de inserción cuando el editor está en esa modalidad.

Apéndice D

Diferencias entre el SV-BASIC y el MSX-BASIC

Los ordenadores Spectravídeo SV318 ySV328 incluyen una especificación de lenguaje que se aproxima mucho a la del MSX-BASIC. Sin embargo, existen ocho diferencias entre los dos. El SV-BASIC maneja las modalidades de pantalla, la anchura de pantalla, las teclas de función, el canal de sonido, el click de las teclas y la impresión de forma ligeramente diferente; también incluye dos mandatos de gráficos que no están incluidos en el MSX-BASIC que son GET y PUT. Veamos estas diferencias en orden:

SCREEN <modalidad>[,<opción>]. Mientras que el MSX-BASIC permite cuatro modalidades de pantalla diferentes, SV-BASIC sólo permite tres. Se especifican de la siguiente forma:

SV-BASIC	MSX-BASIC	Modalidad
SCREEN 0	SCREEN 0	40 X 24 Modalidad de texto
SCREEN 1	SCREEN 2	Modalidad gráficos alta resolución
SCREEN 2	SCREEN 3	Modadlidad gráficos baja resolución

Como puede verse, sólo hay una opción disponible para el mandato SCREEN en SV-BASIC, comparado con el rango de opciones del MSX-BASIC. Esta opción varía con arreglo a la modalidad de pantalla que se esté utilizando.

En modalidad de texto, si la opción es cero, entonces la visualización de las teclas de función desaparece. Si se hace distinta de cero, entonces vuelven a aparecer las teclas de función. Esta última (¡sorpresa!) es la opción asumida por defecto.

En cualquiera de las modalidades de gráficos, la opción se utiliza para seleccionar el tamaño con que se visualizarán los duendes.

0 selecciona duendes sin aumentar de 8 X 8
1 selecciona duendes de 8 X 8 aumentados
2 selecciona duendes sin aumentar de 16 X 16
3 selecciona duendes de 16 X 16 aumentados

El significado exacto de los duendes, 8 X 8, 16 X 16, aumentados y sin aumentar se da en el capítulo de gráficos.

WIDTH [39][40]. En MSX-BASIC, para pantallas en modalidad de texto, se permite cualquier anchura de pantalla comprendida entre uno y cuarenta caracteres. En SV-BASIC, la anchura está limitada a 39 ó a 40 caracteres.

KEY [ON][OFF]. En MSX-BASIC, el mandato KEY se utiliza para alternar la visualización de las teclas de función en la parte inferior de la pantalla. En SV-BASIC, se utiliza la opción de pantalla descrita anteriormente.

SOUND [ON][OFF]. El canal SOUND se puede activar y desactivar en SV-BASIC lo cual no es posible en MSX-BASIC. ¡En el último SOUND se limita a producir sonidos extraños y maravillosos!

CLICK [ON][OFF]. En MSX-BASIC, desactivar el click de las teclas es una de las opciones del mandato SCREEN. En SV-BASIC, el click de las teclas tiene su propio mandato –CLICK ON hace que las teclas suenen, con CLICK OFF las teclas son silenciosas.

PRINT. En SV-BASIC, Vd. puede visualizar cosas satisfactoriamente, en cualquiera de las pantallas, con el mandato PRINT. En MSX-BASIC, PRINT funciona sólo en las pantallas de texto, no en las pantallas de gráficos, y se requiere un sistema más complicado (ver la descripción de la sentencia PRINT en el Capítulo 2 para una descripción completa).

GET (<x1, y1>) – (<x2, y2>), <nombre del array>. En SV-BASIC, cuando Vd. ha realizado un dibujo en la pantalla de gráficos, el mandato GET le permite salvar parte del dibujo en un array. Si se utiliza conjuntamente con PUT, puede volver a colocar, en cualquier zona de la pantalla, la parte del dibujo que Vd. salvó. Es mucho más fácil ver esto con un pequeño programa:

```
10 SCREEN 1
20 DIM A(1000)
30 FOR X = 10 TO 100 STEP 10
40 CIRCLE (X,X),X
50 NEXT X
60 GET (10,10)-(100,100),A
70 PUT (125,100),A,PRESET
80 GOTO 80
```

La línea 10 selecciona la pantalla de gráficos en alta resolución. La línea 20 dimensiona un array llamado A de fondo que sea lo suficientemente grande como para almacenar la parte de pantalla que se va a "tomar". Las líneas 30 a 50 dicen al ordenador que dibuje una serie de círculos empezando con un radio de 10 pixels y centrado en el punto (10, 10); luego con radio 20 pixels y centrado en (20, 20) y así hasta un círculo de radio 100 pixels y centrado en (100, 100). La línea 60 le dice que TOME (GET) una parte del dibujo, empezando en el punto 10, 10 y terminando en el punto 100, 100 y la almacene en A. A continuación, la línea 70 toma A y lo PONE (PUT) en la esquina superior izquierda en la posición (125, 100). PSET es una de las opciones que se pueden utilizar con PUT para variar la forma en que se volverá a obtener el dibujo en la pantalla.

PUT (<x, y>), <nombre del array>[,<opción>]. Es el mandato PUT el que copia la parte del dibujo especificada por GET, en cualquier otra parte de la pantalla. La sintaxis del PUT es muy sencilla –x e y especifican el punto de la pantalla en el que aparecerá la esquina superior izquierda de la paterna que se va a copiar y el nombre del array es el nombre que se le dio a la paterna cuando se utilizó la sentencia GET. Las opciones que se pueden especificar cuando se vuelve a dibujar la paterna en la pantalla son las siguientes:

PSET. Hace que la paterna vuelva a la pantalla exactamente en la misma forma en que se salvó en el array.

PRESET. Hace que la paterna se vuelva a dibujar igual que con PSET, pero con los colores de primer plano y de fondo invertidos.

AND. Combina el color de la paterna con el color de la pantalla de forma que la paterna que se está superponiendo en la pantalla sólo se dibuja si ambos colores coinciden.

OR. La paterna que se superpone se solapa con la paterna que ya está en pantalla, de forma que ambas son visibles.

XOR. Si un pixel de la paterna que se está superponiendo en la pantalla coincide con alguno de los que ya están, entonces se visualiza en el color de fondo. XOR es la opción por defecto.

Aquí hemos intentado explicar lo que hacen las diferentes opciones, no obstante, la mejor forma de comprenderlas es probarlas Vd. mismo. El método más sencillo de hacer esto es volver al programa que hemos utilizado para demostrar la sentencia GET, y reemplazar el PSET de la línea 70 con cada una de las distintas opciones.

OTROS LIBROS SOBRE "INFORMATICA" PUBLICADOS POR

PARANINFO S.A.

Generalidades

ABRAMSON.– Teoría de la información y codificación.
FLORES.– Estructuración y proceso de datos.
GARCIA SANTESMASES.– Cibernética. Aspectos y tendencias actuales.
GOSLING.– Códigos para ordenadores y microprocesadores.
LEWIS y SMITH.– Estructuras de datos. Programación y aplicaciones.
NANIA.– Diccionario de Informática. Inglés-Español-Francés.
PUJOLLE.– Telemática.
SCHMIDT y MEYERS.– Introducción a los ordenadores y al proceso de datos.
URMAIEV.– Calculadores analógicos. Elementos de simulación.

Lenguajes

BELLIDO y SANCHEZ.– BASIC para Maestros.
CHECROUN.– BASIC. Programación de microordenadores.
DELANOY.– Ficheros en BASIC.
GALAN PASCUAL.– Programación con el lenguaje COBOL.
GARCIA MERAYO.– Programación en FORTRAN 77.
HART.– Diccionario del BASIC.
LARRECHE.– BASIC. Introducción a la programación.
MARSHALL.– Lenguajes de programación para micros.
MONTEIL.– Primeros pasos en LOGO.
ROSSI.– BASIC. Curso acelerado.
SANCHIS LLORCA y MORALES LOZANO.– Programación con el lenguaje PASCAL.
WATT y MANGADA.– BASIC para niños.
WATT y MANGADA.– BASIC avanzado para niños.
WATT y MANGADA.– BASIC para niños con el microordenador DRAGON.

Aplicaciones e Informática Profesional

ANGULO.– Curso de Robótica.
ANGULO.– Robótica práctica.
ANGULO.– Prácticas de microelectrónica y microinformática.
ASPINALL.– El microprocesador y sus aplicaciones.
BANKS.– Microordenadores. Cómo funcionan. Para qué sirven.
BELLIDO.– Amaestra tu DRAGON. Curso de programación BASIC para microordenador DRAGON.
BELLIDO.– Cómo programar su Spectrum.
BELLIDO.– Cómo usar los colores y los gráficos en el Spectrum.
BELLIDO.– KIT de gráficos para Spectrum.
BELLIDO.– Enciclopedia del Spectrum. Tomo I.
BELLIDO.– Spectrum. Iniciación al Código Máquina.
BELLIDO.– ZX81. Curso de programación en BASIC.
ELLERSHAW.– Las primeras 15 horas con el Spectrum.
ERSKINE.– Los mejores programas ZX Spectrum.
ESCUDERO.– (Centro de Investigación UAM-IBM).–Reconocimiento de patrones.
FERRER.– Programas en BASIC.
GAUTHIER y PONTO.– Diseño de Programas para sistemas.
HARTMAN, MATTHES y PROEME.– Manual de los Sistemas de Información. 2 tomos.
LUCAS, Jr.– Sistemas de información. Análisis. Diseño. Puesta a punto.
MARTINEZ VELARDE.– El libro de Código Máquina del Spectrum.
MONTEIL.– Cómo programar su Commodore 64. Tomo I.
PANNELL, JACKSON y LUCAS.– El microordenador en la pequeña empresa.
PLOUIN.– IBM-PC. Características. Programación. Manejo.
WILLIAMS.– Programación paso a paso con el Spectrum.

MSX. Programación básica

La intruducción del estándar MSX es, sin lugar a dudas, el hecho más importante en la historia de la computarización en el hogar.

Este es un libro de iniciación para todos los nuevos usuarios del MSX. Abarca desde los principios más elementales, hasta las complejidades de la programación en BASIC avanzada, pasando por técnicas más complejas, como gráficos y música. Incluye comentarios y explicaciones teóricas de cada programa.

Todo ello hace de este libro una obra muy útil tanto para principiantes como para programadores avanzados que deseen profundizar en el revolucionario MSX.

MSX sirve para todos los ordenadores personales de origen japonés y también para algunos Philips.

INDICE EXTRACTADO: Presentación del MSX. Cómo programar en MSX-BASIC. Cómo trabajar con números. Cómo interactuar con sus programas. Música y sonido en el MSX. Gráficos en MSX-BASIC. Apéndices: Funciones. Códigos de error y mensajes. Teclas de control para el editor de pantalla completa. Diferencias entre el SV-BASIC y el MSX-BASIC.



Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1407-1